REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)* 31-01-2008	**2. REPORT TYPE** Final Report	**3. DATES COVERED** *(From – To)* 17 July 2006 - 04 February 2009

4. TITLE AND SUBTITLE Negotiation based deconfliction in air-traffic control	**5a. CONTRACT NUMBER** FA8655-06-1-3073
	5b. GRANT NUMBER Grant 06-3073
	5c. PROGRAM ELEMENT NUMBER 61102F
6. AUTHOR(S) Michal Pechoucek, David Sislak, Premysl Volf, Stepan Kopriva, Jiri Samek	**5d. PROJECT NUMBER**
	5d. TASK NUMBER
	5e. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Czech Technical University, Agent Technology Group, Gerstner Laboratory -- CTU, FEE-K333, Technicka 2, Prague 6 166 27, Czech Republic	**8. PERFORMING ORGANIZATION REPORT NUMBER** N/A
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) EOARD, Unit 4515 BOX 14, APO AE 09421	**10. SPONSOR/MONITOR'S ACRONYM(S)**
	11. SPONSOR/MONITOR'S REPORT NUMBER(S) Grant 06-3073

12. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This report results from a contract tasking Czech Technical University as follows: The Grantee will investigate development of a model and software prototype an autonomous embedded UAV air traffic deconfliction demonstration..
This project leverages results of previously implemented project (funded as FA8655-04-1-3044-P00001) that resulted in an A-globe multi-agent system deployment in the domain of air-traffic control. Within the proposed project we intend to integrate the previously implemented technology with state-of-the-art deconfliction methods and provide empirical evaluation of various approachesand to generalize the deconfliction and related data-collection technology to allow deconfliction with non-cooperative aerial objects.

15. SUBJECT TERMS

EOARD, Multi Agent Systems, Command and Control, Computer Modelling

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18, NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**	SAR	148	JAMES LAWTON Ph. D.
UNCLAS	UNCLAS	UNCLAS			**19b. TELEPHONE NUMBER** *(Include area code)* +44 (0)1895 616187

Project extension of the FA8655-06-1-3073 contract: Final Report January 2008

this is a final project report to extension of the FA8655-06-1-3073 contract and it provides technical information about the work on the project

**Michal Pěchouček^{PI}, Přemysl Volf,
David Šišlák, Štěpán Kopřiva**

Agent Technology Group, Gerstner Laboratory,
Czech Technical University in Prague

Contents

1 Executive Summary

This document provides a final report of the FA8655-06-1-3073 project extension. The FA8655-06-1-3073 dealt with deployment of multi-agent modeling for collective flight modeling and development of distributed, agent-based collision avoidance methods. Properties of the methods were experimentally tested within the AGENTFLY multi-agent system. The whole technology and the progress of work has been described in the final report of the FA8655-06-1-3073 project that is delivered together with this report. Hereafter we will refer to this report – final report of *FA8655-06-1-3073-extension* and to the original report – final report of *FA8655-06-1-3073-main*.

The final report of FA8655-06-1-3073-main provides:

- detailed specification of the domain where multi-agent techniques has been applied,
- flight modeling and time-constrained way-point flight plan planning algorithm avoiding ground surface and no-flight zones,
- description of the AGENTFLY prototype architecture including real-time visualization component, remote web-based access and its integration with various external data sources,
- multiple operator agent interface providing human-system interface allowing real-time control of UAAs,
- multi-layer collision avoidance architecture allowing simultaneous combination of cooperative and non-cooperative collision avoidance methods,
- describes three implemented cooperative collision avoidance algorithms: *rule-based*, *iterative peer-to-peer* and *multi-party collision avoidance*,
- provides description of non-cooperative collision avoidance based on the dynamic no-flight zones,
- collective flight coordination architecture used for the synchronization of the group of UAAs supporting flight in the formation as well as outer group-to-group negotiations
- basic set of the empirical experiments comparing all cooperative methods together, non-cooperative no-flight zone method comparison to optimal proportional navigation algorithm
- specialized demo cases demonstrating benefits of the multi-party algorithm and validating the concept of multi-layer collision avoidance architecture where the agent controlled airplanes operate in the area with civil traffic,
- complex combat scenario used as a testing case for the mix of all features provided by AGENTFLY in a very complex mission.

The main project has been extended during its performance by the following specific tasks:

- a theoretical study of the properties (and worst case scenarios) of the used deconfliction mechanisms,
- thorough scalability tests providing the empirical properties of the used technology and comparison with the latest state-of-the-art,
- theoretical foundation for an extension of the investigated and developed deconfliction architecture towards team-oriented action and operation in adversarial environment

The work has been organized into four research targets. The indispensable research review, prerequisite for the theoretical research work has been performed in the RT4 in the section 2. The theoretical work, embedded in the RT1 has been performed subsequently in the section 4. The scalability tests, that were grouped in the RT2 in the section 5, were implemented at the same time as the theoretical work was carried out. In the RT3 we studied and designed deployment of AGENTFLY technology in the collective flight concepts and in the adversarial and combat unit deployment scenarios described in the sections *Collective flight* and *Complex combat scenario* in the main report. All research targets have been fully achieved.

2 Review of Existing Algorithms for Air Traffic Collision Avoidance

The air traffic collision avoidance problem is a problem how to solve collisions of the airplanes arising in the *free flight* concept [9]. Different methods solving this problem are described in this section. We use multi-agent simulation system AGENTFLY described in the main report to solve collision avoidance problem. For the formal verification and empirical experiments is used iterative peer-to-peer collision avoidance (IPPCA) algorithm described in the section 3.

There are various collision avoidance metrics that compare functionality, efficiency, stability, speed and other characteristics for every algorithm. These metrics suggest how appropriate is a given algorithm for a given collision avoidance problem. Krozel proposes the following metrics in [6]. The metrics measure properties of the system defined as a sum of the properties of all airplanes. The *stability* compares number of collisions, when airplanes fly along their nominal trajectory, to number of detected collisions after execution of the conflict resolution manoeuvres of all airplanes. This metrics is suitable for algorithms that are not able to solve all collisions and number of collisions after algorithm execution is comparable to a number of collisions before algorithm execution. This metric is unsuitable for advanced algorithms (IPPCA algorithm, Hill's algorithm 2.1), where the number of collisions after execution of the conflict resolution manoeuvres is zero for majority of the scenarios. Total number of collisions and minimal separation violations is measured in this case. The *efficiency* is the degree to which an aircraft can fly its nominal trajectory. Any deviation from the nominal for conflict resolution results in an additional operational cost (eg. fuel consumption, time restrictions, length of the trajectory). The *airspace complexity* or dynamic density is not defined precisely. However, number of airplanes in specified airspace is key factor. There are also many other factors like velocity and space constrains, proximity to other airplanes etc. We measure number of collisions, minimal separation violations and system efficiency in the experiments 5.

To show described properties of the algorithms, we study the algorithms theoretically, perform empirical behavior of the flight simulation or use some other AI methods (such as model-checking). The formal verification of collision avoidance methods deployed for the complete collision avoidance problem is a complex and difficult task. That is why researchers formally study only subtasks of the collision avoidance problem.

To study collision avoidance problem as a system is suitable to use multi-agent simulation. It allows to run complex scenarios and study nondeterministic behavior caused by asynchronous events during the simulation. The verification is done by huge number of experiments to receive sufficient statistical data.

Tomlin *et al.* [10] solve a problem of two noncooperative airplanes. They start with the differential equations in the game theory used for intelligent vehicle systems. They determine safe and unsafe regions for an airplane and the airplane can operate only on the boundary of its safe region to optimize trajectory and other criteria. The implicit switching control law derived from the solution to the game may be modeled by finite automata with differential equations associated with each state, and the resultant hybrid system is safe by design.

Han *et al.* [3] use proportional navigation-based optimal collision avoidance algorithm. The optimal proportional navigation algorithm tries to keep a predefined safe distance between an airplane and an obstacle (non-cooperative one tracked on its on-board radar). This is achieved by directing the airplane towards the edge of the safety zone and an appropriate acceleration. If the airplane gets into such configuration with the obstacle in which no future collision exists, the airplane control algorithm switches its mode from deconfliction to navigation towards the destination point. This algorithm is able to avoid only one airplane/obstacle.

Krozel compares three different approaches to conflict resolution – one centralized and two distributed methods [6] verified by simulation. Centralized strategy emphasizes stability of the system by suppressing domino effect, but the system efficiency degrade with increasing traffic density. Decentralized algorithms use myopic and look-ahead strategies. The myopic strategy always selects the most efficient solution. If this manoeuvre creates a new conflict, it is resolved in the next cycle. The look-ahead strategy also takes the most efficient solution, but takes into consideration stability as well. It can select less efficient manoeuvre that does not decrease stability so much as the most efficient one.

Agogino *et al.* [1] solve problem of the management of the air-traffic in USA, which is so extensive that is not able represent as an agent each aircraft. Agents represent cells of the airspace called ”fixes”. Each agent is responsible for each airplane going through its fix. Number and position of agents is predefined. Each agent uses evolutionary algorithms to improve its ability to control airplanes. Complex solution is composed from partial solution of the agents. Proposed concept is tested in the FACET system and show 40% increase in performance.

Kosecka *et al.* use potential and vortex fields to solve collision up to four aircrafts [5]. Generalized overtake and head-on manoeuvres may solve all two-aircrafts collisions. For more airplanes they propose manoeuvre called roundabout. The results are verified by simulation of the algorithm.

Holdsworth *et al.* [4] models an escape trajectories generated by different algorithms using Brisbane Model for the simulation. It uses tesselated space with aircraft's position represented as one tile. This model with probability and risk calculation is used to identify cases where algorithms fail for further analysis.

Wangermann *et al.* [11] use principled negotiation to effectively coordinate distributed optimization in both constrained and unconstrained situations as extension of the centralized system.

2.1 Hill's Algorithm

We use Hill's algorithm as a reference to the IPPCA algorithm. Selected *perpendicular flows* scenario in the section 5.3 and *circles* scenario in the section 5.4 show comparison of the algorithms.

Hill *et al.* uses satisficing game theory, the concept based on the dual social utility, which is composed of two parts – selectability and rejectability. The *selectability* characterize effectiveness of the solution in reaching the goal regardless the cost of this solution. The *rejectability* describes amount of resources, which are consumed to reach selected solution. The social value is normalized to have mathematical structure of the mass function (but with different semantics). In the multi-agent system, mass function is defined as multivariate mass function containing selectability and rejectability for each agent. Structure of the mass function permits to define independence and conditioning similar to probability theory. This definition allows to model *situational altruism*, where agents are not pure altruists. The agents still follow strategy optimizing their utility, but they are able to concede some of their preferences to another agents, if this concession will lead to reasonable gain of the utility over all agents.

In the Hill's implementation all airplanes fly at the same altitude and at the same constant speed. The simulation is discretized (currently 1 second intervals) and each step is selected one of 5 possible heading changes (-5; -2.5; 0; 2.5; 5 degree). Each airplane ranks others airplane considering distance, delay, flight. Airplanes are consecutively divided into subgroups in the following order:

- airplanes within fifty miles range and airplanes outside fifty miles range
- each group is ordered by accumulated delay
- airplanes with the same delay are divided according to how long they are in the air (longer time has higher priority)

Each airplane builds acyclic graph representing influence flow from higher ranked airplanes to lower ranked ones.

Each airplane compute its rejectability utility for each possible direction. Each airplane with higher priority will leading to a collision will affected utility by predefined weight (collisions have higher value than near misses). Selectability utility is influenced by the difference between selected heading and heading to the destination. Each airplanes have to take into account selectability utility of the higher ranked airplanes to get social utility. Both utilities are normalized.

Hill have developed two models - *full model* and *simplified model*. An airplane in the full model tries to compute complete selectability of the higher ranked airplanes using its local incomplete knowledge. This model improve overall performance, but is very complex. In the simplified model are airplanes divided into five groups according to the possible heading changes. Number of airplanes in each group is taken as a weight for the group.

Final decision is made according to whether the agent is risk averse or risk seeking. Risk averse agents selects option with the lowest rejectability utility and risk seeking agent selects option with the highest selectability utility. Hill have selected option with the highest difference between selectability and rejectability utility for each airplane.

3 Extended Iterative Peer-to-Peer Collision Avoidance Algorithm

The iterative peer-to-peer collision avoidance algorithm (IPPCA) is a domain independent algorithm that is used for collective collision avoidance by negotiating over domain dependent deconfliction manoeuvres. The IPPCA algorithm is utility-based avoidance mechanism providing solution for a pair of airplanes. Collisions of more than two airplanes at the same time (multi-collisions) are solved iteratively.

3.1 Original Algorithm

Original version of the IPPCA algorithm is an extension of [12]. First, the participating airplanes select the master and the slave entities for the detected collision (usually the first entity who identifies a collision is regarded as a master entity). In the Figure 1 there is negotiation flow between both participants.

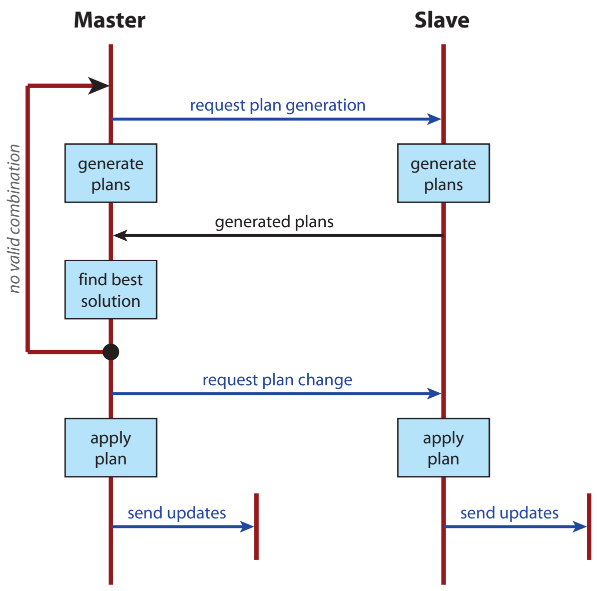


Figure 1: The negotiation used by IPPCA algorithm

Each agent generates a set of plans using defined manoeuvres. All pre-defined manoeuvres also allows to apply different level of changes depending on the *generation parameter*. The result of the application of the manoeuvre is a new changed flight plan including utility value for this plan. This new plan is checked for the safety zone violation against all other airplanes in the area (excluding second airplane in the pair). If there is safety zone violation with any of these airplanes, new plan is excluded. Otherwise it is put into list of generated manoeuvres. The generation parameter during the generation process is used to get wider range of solutions in the situations when the solution is not found using flight plans with smaller changes.

The utility function is used to include the aircraft's intention to the proposed solutions of the conflict. The utility value is evaluated as weighted sum of the utility function parts. Depending on the configuration there can be components taking into consider total length of the flight plan, time deviations for mission waypoints, altitude changes, curvature, flight priority, fuel status, possible damage, type of load. Relative utility value exchanged between planes is computed as quotient of new flight plan to original flight plan. Lower value of utility function suggest more preferred deconfliction manoeuvre.

There are 7 parameterized manoeuvres used in the current version of IPPCA algorithm: straight manoeuvre (no change to the flight plan), turn right, turn left, turn up, turn down, speed up and slow down manoeuvre.

The best possible deconfliction manoeuvres is identified by a variation of the *monotonic concession protocol* (MCP) [12] described in the appendix A. The monotonic concession protocol is a simple protocol developed by Zlotkin and Resenschein for automated agent to agent negotiations (see below). Instead of iterative comparison of the most preferred maneuvers of each party, the complete ordered set o flight plans (and labeled by the utilities) are generated and sent back to the master airplane. When the master entity generates its own plans and receives plans from the slave entity, it tries to combine all plans together. The collision solution is then selected from cartesian product of the generated plans from both participants. These candidates for solution are ordered in increasing manner by product of utility quotients of flight plan pair. The order of the candidates by the sum of their utilities optimize the social welfare. Each solution candidate is tested for a collision between airplanes from this pair. If there is no collision between participants, candidate is selected as collision solution. When there are more solution pairs with the same sort value without collision, the the final solution is selected randomly from these. The slave entity is notified about selected flight plan. This approach turns out to save substantial amount of communication and consequently makes the solver more likely to provide a solution prior a possible collision.

If there is no collision-free pair in the cartesian product, it is necessary to generate different wider flight plans and the request for generation of the new set of possible flight plans with greater generation parameter for manoeuvres is sent to slave entity. The master entity generates its new set of flight plans as well. This new plans are added to flight plans from previous round of generating and the master will repeat to selection of the solution among all flight plans.

Original implementation of the IPPCA continues in increasing generation parameter to get wider manoeuvres until solution is found or the CSM timeout elapses. CSM - Collision Solver Manager is module in the AGENTFLY multi-layer architecture (see the main report in the Section *Multi-layer collision avoidance architecture*) that manage collision avoidance modules. For each collision is selected appropriate collision solver (e.g. IPPCA solver). The solver receives predefined amount of time to solve the collision. When timeout elapses and no solution is found, both airplanes in the pair keep their original flight plans. It can be set, whether collision is considered as solved and no new negotiation is initiated or the collision should be detected again and new round of the negotiation should run. This process can lead to continuous negotiation without any change of the flight plan and collision of the airplanes. This situation can occur in scenarios with extreme density of the airplanes at same time. Modification for these scenarios is introduced in subsection 3.3.

3.2 Extension with tendencies

We have developed extension of the original IPPCA algorithm for the most of the possible scenarios (excluding some specific scenario with extreme density, where results can be worse than using original algorithm). This extension reduces number of iterations of negotiations, reduces the necessary communication flow among airplanes and makes final flight plans more simple (lower number of final segments). This advantage is allowed by slightly longer total trajectory of the flight plans.

To reduce the number of iterations, it is applied following restrictions to the applicable evasion manoeuvres. All seven defined manoeuvres are divided into four groups: (i) *horizontal* – right and left evasion, (ii) *vertical* – climb up and descend down evasion, (iii) *velocity* – fly faster and slow down evasion and (iv) *straight* – holding only the straight evasion. When the multi-collision is detected, it has to be solved by several iterations of the IPPCA between airplane pairs included in this multi-collision. For each airplane and each application of the evasion manoeuvre is saved type of the evasion manoeuvre. Once the manoeuvre is applied from some group, it is not allowed to use another type of the manoeuvres from this group until the multi-collision is solved (e.g once it is used slow down manoeuvre for some airplane, the fly faster manoeuvre can't be used for this airplane until the multi-collision is solved). This extension reduces cases, when original algorithm leads to series of application of the opposite manoeuvres from same group (turn left, right, left, right, ...).

3.3 Extension with Near Misses Optimization

We have developed extension of the IPPCA algorithm (can be used together with tendencies) for the specific scenarios or for the scenarios with extreme density of airplanes, where CSM timeout could elapse before any solution respecting minimal separation is found. Original algorithm can cancel negotiation (when CSM timeout elapses) and no changes to flight plans are applied, which can lead to the collision of the airplanes. To avoid such situations, it can enabled feature minimizing possibility of a the collision using airspace where minimal separation is violated, but still respecting collision zones (zones where collision of the airplanes is simulated).

In this extension is set maximum size of the generation parameter during negotiation (this parameter should be set to such value that does not limit possible solution and at the same time generation of all plans have to finish before CSM timeout). When the algorithm reaches this maximum and no solution respecting minimal separation is found, it tries to search for a solution where the minimal separation is violated, but the violation is minimized and the collision zone (where real collision is expected) is respected.

Implementation of this extension is optimized to reduce computation load, to spread out computation over whole negotiation and not to make huge computation when maximum of the generation parameter is reached.The implementation of the algorithm differs form the original algorithm in the following:

- **during the phase when of the the new manoeuvres on the master and slave side are generated for the first time (generation parameter equals 1)**

  Manoeuvre is added to the generation set even when violates minimal separation (but not collision zone) with other airplanes. To each manoeuvre is assigned *relative safety zone violation* (RSZV) value ($<0;(1\text{-relative collision zone size})>$, zero is no violation to minimal separation, 1-relative collision zone size means collision of the airplanes) as a maximum violation over all airplanes (excluding other airplane in the pair).

- **during the phase, when the master checks for possible solution from combinations of the generated plans from master and slave**

  The *combined RSZV* for the combination of the master's and slave's plan is defined as the maximum of the master plan's RSZV, slave plan's RSZV and minimal separation between the master and the slave.

  The *minimal RSZV* is the minimum of the combined RSZV over all combinations (initially set to 1).

  If the solution is found (combined RSZV==0), extended algorithm continues in the same way like the original algorithm (solution selection and application). If the solution is not found, all combinations created from plans with RSZV$\leq$minimal RSZV are checked and new minimal RSZV is set.

- **during the phase when of the the new manoeuvres on the master and slave side are generated during next iterations (generation parameter equals 1)**

  Manoeuvre is added to the generation set if the manoeuvre's RSZV$\leq$minimal RSZV (No need to add manoeuvres with bigger safety zone violation than it has been reached in previous iterations).

- **during the phase, when the master checks for possible solution from the combinations and the maximum of the generation parameter is reached**

If the solution is not found (combined RSZV>0), it is not possible to continue to next iteration. The master combines all plans (from both master and slave) with RSZV≤minimal RSZV. As a solution is selected combination with minimal combined RSZV. In case there are several combinations with the same combined RSZV, it is selected combination with best aggregate utilities (similarly to original algorithm).

4 Theoretical Analysis

We show theoretical properties of the IPPCA algorithm described in the section 3.1. We formalize the IPPCA algorithm and select extreme *landing scenario* with a high density of airplanes in a limited area. We show theoretical properties, estimations and restrictions in this scenario. The theoretical have been published in [7].

4.1 Landing scenario

Landing scenario contains n airplanes forced to fly from their starting positions through the single landing point and follow straight part common to all airplanes simulating landing on an aircraft carrier, see Figure 2. All airplanes fly at the same altitude and they cannot manoeuvre and avoid collisions by changing their altitude. The airplanes have to arrange their times of arrival in such a way, that no more than one airplane flies through the landing point at a time.

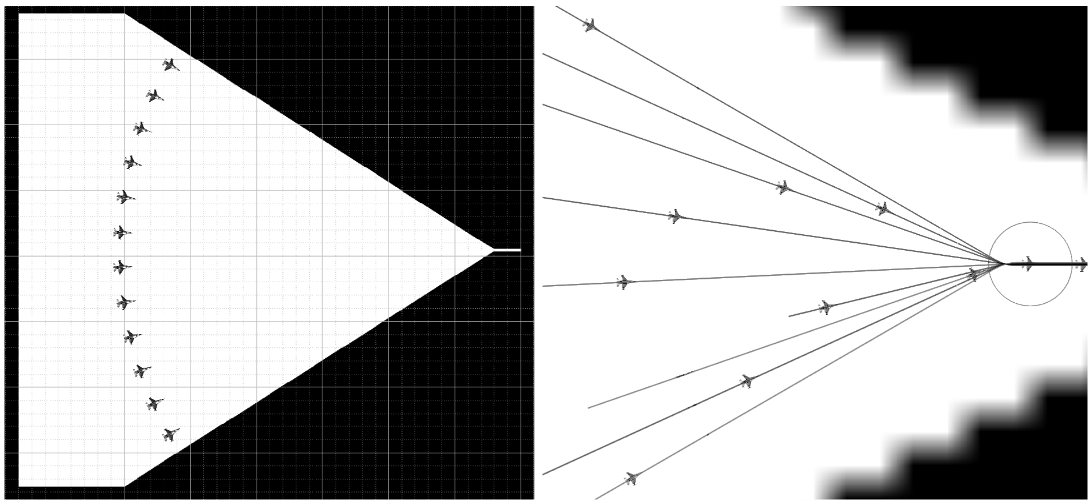


Figure 2: Landing scenario. Airplanes arrange their time of arrival to the landing point. Circle around the airplane represents the safety zone.

By closer look at the scenario, we can see that in order to avoid collisions behind landing point, all airplanes have to fly at the same speed to the landing point and then keep their speeds until they reach the final way-point. The aircrafts will avoid collisions by careful arrangement of their landing point arriving times and by keeping minimal distances from other aircrafts equal at least to the safety zone size. Without loss of generality, we suppose only speed ups and slow downs are allowed. Any other manoeuvre that would change airplane's trajectory would result in different time of arrival to the landing point and therefore it wouldn't bring any additional possibility how to avoid collision.

4.2 Assumptions and Objectives of the Model

We have built formal model of the landing scenario. This model should keep properties and complexity of the simulated scenario, but it should also provide us with the possibility to use formal methods to prove convergence.

The only important aspect of the model is the time of arrival of each airplane to the landing point, thus all flights can be modeled as a intervals on the time axis representing airplane's safety zone, see Figure 3. As a simplification we regard starting position of each airplane only as time needed to fly to the landing point using its original speed. Note that we loose information about position of the airplanes. We do not know when and with the first collision occur in this model, so the closest neighbor at the time axis does not need to be airplane with the soonest collision. Only information is whether or not collision will occur.

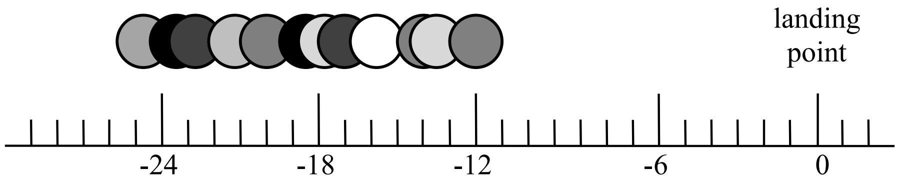


Figure 3: Tunnel scenario modeled as time axis. Airplanes are on their starting positions. Overlapping airplanes will have a collision in the starting point.

We assume that at the beginning speed of all airplanes is identical and they will fly through the landing point at this speed. Speed up and slow down manoeuvres are transformed to adjustments of aircraft's time of arrival to the landing point (time jumps). Thus speed up manoeuvre does not mean, that airplane will increase its speed, but that it will arrive to the landing point sooner maintaining its original speed. This can be done by speeding up for necessary time to gain time and then slow down to the original speed. Slow down is defined in similar way. Denote *shift forward* manoeuvre, when the flight time to the landing point is decreased and *shift backward* manoeuvre when the flight time to the landing point is increased. We assume number of the shifts in either way is not limited. Size of this zone is defined as the time needed to fly through the safety zone of an aircraft at the original speed.

Utility function is defined as local optimization, this means that utility value is optimized for each deconfliction separately. Every airplane remembers its actual position at the beginning of each deconfliction and this position generates the best value of the function. Utility value is getting worse with (time) distance from actual position in the same way in both directions. Thus every airplane tries to minimize necessary change from its actual position. If several minimal choices are possible, random solution is selected.

It is guaranteed that airplanes trust each other and offer true and reliable information. We also assume that each airplane has access to the information about flight plans of all other airplanes. We assume communication is parallel, but synchronous. Thus only negotiation of one pair can be performed at a time. This prevents multiple changes of the flight plans and possible reverting some of them. We assume communication is reliable.

4.3 Formal Proof of Convergence

Definition 1 (First (last) airplane) *First (last) airplane is the airplane with the earliest (latest) time of arrival to the landing point. If there are several airplanes with the earliest (latest) time, all of them are considered as the first (last) airplane.*

Definition 2 (State of the problem) State *of the deconfliction problem is defined by positions of all airplanes at the time axis.*

Definition 3 (Step) *One* Step *of the algorithm is the application of utility based negotiation to pair of aircrafts and then the aircrafts perform appropriate changes (shifts) to their positions. One step transforms one state to another.*

Definition 4 (Cycle) *Let's define* cycle *as a sequence of the steps, where the state of the deconfliction problem is the same before and after the performance of these steps.*

Definition 5 (Restricting neighbors) *All airplanes are tagged by either restricting or non-restricting tag for each deconfliction. Suppose an airplane A is trying to shift from its original position to the new position p (to solve the collision c). Airplane X is marked as restricting for the airplane A, collision c (its time) and the new position p, if airplane A can not solve collision c by moving to the position p because collision with X would arise sooner then c. Otherwise (if there is no collision taking place sooner than collision c, including no collision at all) the airplane X is marked as non-restricting. Desired shift to the new position is allowed only if no restricting airplane exists for this position.*

Definition 6 (Constants and variables) *Constants (C) are same all the time, variables (V) can vary. Denote*

- *sz (C) as a size of safety zone measured in time units.*
- *d (V) as a distance (in time) between airplanes on the time axis, ie. difference in arrival times of airplanes to the landing point.*
- *D (V) as a distance (in time) between the first and the last airplane.*
- *ms (C) as a manoeuvre step, minimal size of the shift forward or backward. Airplanes are shifting only in multiplies of this value.*
- *$ts(D) = \frac{D}{ms}$ (V) as a total number of positions between the arrival to the landing point of the first and the last airplane.*
- *$s(n, D)$ (V) as a number of all possible states between actual first and last airplane. $s(n, D) = ts(D)^n$*

Lemma 1 *The position of the first (last) airplane cannot be shifted backward (forward).*

Proof. Airplane shifts forward or backward in time only as a result of a deconfliction. Conflict for the *first* airplane can arise only with airplane with later time of arrival (from the definition).

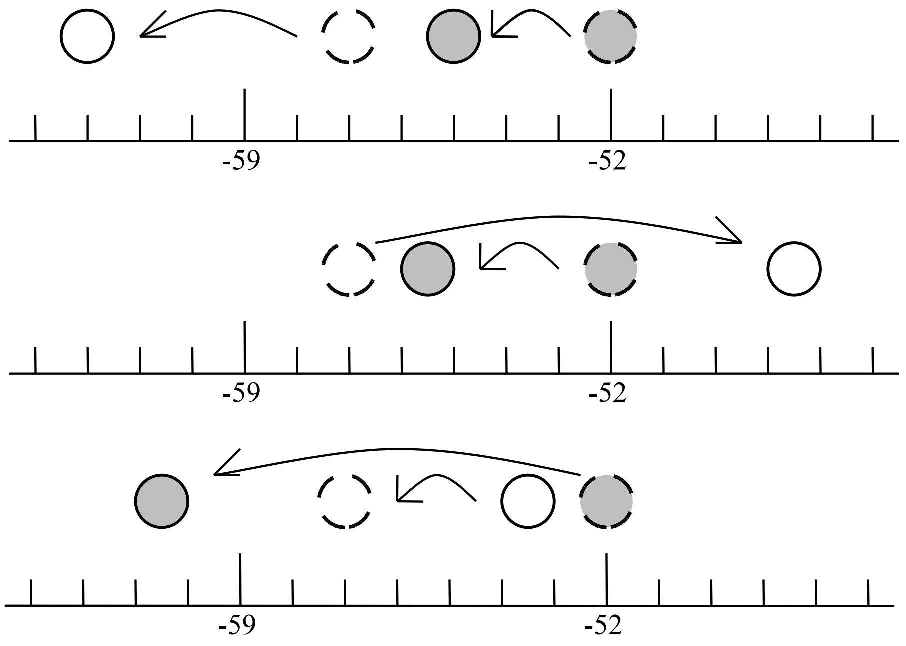


Figure 4: Deconfliction of the *first* airplane (grey). Original positions are drawn as dashed lines.

Suppose that the *first* airplane will shift backward. Figure 4 shows all possible changes of the airplanes[1].

In the first case, there is no reason for the *first* airplane (grey) to shift backward, because the new utility value will be worse than utility value for its original place.

In the second case, airplanes switch their positions and second (white) airplane gets in front of the grey one. Thus the white airplane is marked as new *first* airplane and thus position of the *first* airplane is shifted forward.

In the third case, airplanes are switched, but white is still behind the original position of the *first* airplane. Better change for the *first* airplane would be to shift forward instead of backward, because it leads to the smaller change from its original position.

Also situation where two airplanes (both *first*) are solving their collision can appear. Although even in this case there is no reason to shift both airplanes backward. This situation is analogous to the first case. □

Lemma 2 *Assume each airplane solves its soonest collision first, then cycle cannot arise.*

Proof. We prove this lemma by contradiction. Assume that cycle exists. This cycle cannot contain any collision free state, because algorithm would immediately stop and cycle wouldn't be formed. Therefore all states in the cycle contain a collision.

Each step of algorithm is a result of solving some collision. Denote c as a first collision being solved in the first state of the cycle. Consequently after solving collision c and moving to the

[1]size of the safety zone is defined by larger lines at the time axis. It has size of 7 in this Figure.

second state collision c doesn't exist any more. No collision can arise sooner or in the same time as a result of a deconfliction (as a property of the utility based algorithm). No state after second state can contain collision c and therefore the first and the last state is not same. □

Implication 3 *D has to be increased after $(2n-1) \cdot s(n,D)$ steps by at least ms.*

Proof. As a result of Lemma 2 after exhausting all possible states ($s(n,D)$ steps) without solving collisions either the *first* or the *last* airplane has to shift. Lemma 1 proves that the *first* airplane can shift only forward and the *last* airplane can shift only backward. Thus the distance between these airplanes D has to be increased.

How long does it take to increase D for at least ms? Suppose only shifting of the *first* airplane forward after $s(n,D)$ steps. Let's mark the *first* airplane as airplane A. The position of the *first* airplane can be shifted forward in two different ways. Firstly, airplane A shifts forward for the minimal step ms, therefore the implication holds.

Secondly, other airplane (airplane B) shifts in front of the airplane A and is marked as new *first* airplane. Distance between the original position of first airplane A and the new position of first airplane B can be lower than ms and thus insufficient. After n takeovers of the leader, at least one airplane has to be in lead twice and therefore this airplane moved at least for ms, which shows that implication holds.

Let's allow also shifts of the *last* airplane backward. Position of the last airplane can change at most $n-1$ times without increasing the D by ms.

Since each shift of the first/last position is performed after maximum of $s(n,D)$ and there can't be more than $n-1$ on each side without increasing the D for ms, the lemma is proved. □

Lemma 4 (Maximal separation) *As a result of the single deconfliction, the distance between any two neighboring (on the time axis) airplanes cannot be increased to value greater than*

$$d \leq 2 \cdot sz + ms$$

Proof. Assume two airplanes are solving their collision. Algorithm starts by each airplane generating its possible future positions. Several situation can occur.

In the first case, new positions are not restricted by any other aircraft and changes can be applied. After the deconfliction the distance between airplanes will stay $d < sz + ms$.

In the second case, an aircraft restricting its movement at one side exists therefore airplanes will use space on the other side to solve the collision and the distance will stay $d < sz + ms$.

In the third case, there are restricting aircrafts at both sides of both airplanes, see Figure 5. Two black circles at the Figure shows restricting aircrafts closest to the pair of airplanes in collision.

Distance between restricting aircrafts is lower than $3 \cdot sz$ and thus there is not enough space to solve deconfliction in between these aircrafts without arising collision with one of them. The situation is solved by extending the shift of the airplane in the conflict to reach position behind restricting aircraft(s). You can see two different cases at the Figure 5.

In the first situation, the distance between black aircrafts is bigger than $2 \cdot sz + ms$, thus one of the airplanes (grey in our case) will jump over the restricting aircraft. Now white airplane has

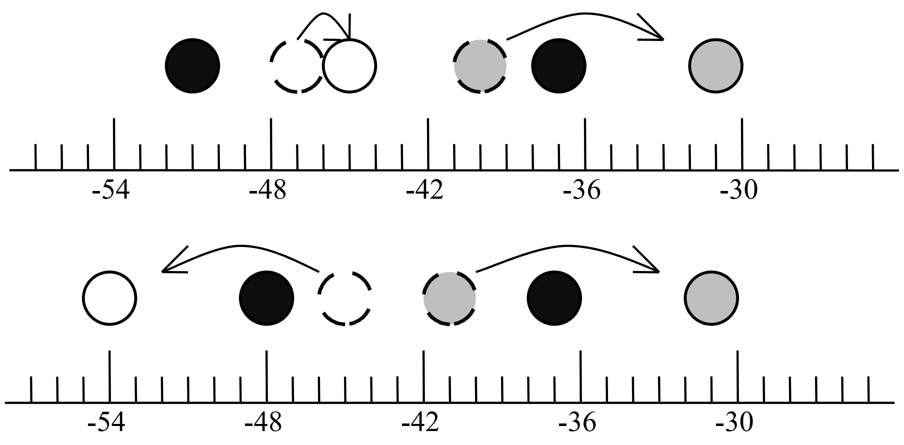


Figure 5: Figure shows deconfliction of the airplanes (grey and white), that are restricted by another (black ones). Deconflicting airplanes cannot shift their position closer than sz to any black airplane.

enough space to find the best position between restricting aircrafts. It will hold minimal separation of sz between white and black airplanes and thus maximal distance between any two airplanes is not increased over $2 \cdot sz + ms$.

In the second situation, no airplane from pair of airplanes in the conflict can stay between black aircrafts. But this also means that distance between restricting aircrafts is not bigger than $2 \cdot sz + ms$. Otherwise there would be enough space for one airplane to stay between them. $\square$

Implication 5 *Minimal distance D ensuring collision free state (where distance between any two airplanes is at least sz) is*

$$D_{min}(n) = (n-2)(2 \cdot sz + ms) + sz$$

Optimal (shortest) distance D, to reach collision free state, is

$$D_{opt}(n) = (n-1) \cdot sz$$

Proof. Collision does not exist between any two airplanes, if the distance between them is at least sz. Lemma 4 says, that distance between any two airplanes cannot be increased to distance greater than $2 \cdot sz + ms$. Thus minimal distance D, that guarantees the distance between any two airplanes to be at least sz, is $D_{min}(n) \leq (n-2)(2 \cdot sz + ms) + sz$.

Distance between any two airplanes have to be at least sz. Thus minimal space to hold all airplanes without collisions is $D_{opt}(n) = (n-1)sz$. $\square$

Theorem 6 (Convergence) *Collision free state for all airplanes is reached within finite amount of algorithm steps.*

Proof. Implication 3 shows that D has to be increased after finite amount of steps for at least nonzero distance. Implication 5 proves that $D_{min}(n)$ is sufficient to collision free state for all airplanes. Thus all collisions are solved within finite amount of the algorithm steps. $\square$

Theorem 7 (The worst case estimation) *Collision free state is reached in maximum*

$$acs(n, D) \leq \sum_{i=D}^{D_{min}(n)} (2n-1)(\frac{i}{ms})^n$$

steps, where D is the distance between the first and the last airplane in the initial state and sum step is ms.

Proof. $(2n-1) \cdot s(n, D) = (2n-1) \cdot ts(D)^n$ steps is needed at maximum to increase D at least for ms. This number of steps has to be added for every change of distance (using the step ms) between the first and the last airplane from the initial D to the D_{min} when collision free state is guaranteed. □

4.4 Estimations and Restrictions

We proved convergence of the problem in previous section. In this section, we will make further analysis. We specify distance of the airplanes from the landing point necessary to solve all collisions, and prove convergence under the assumption that a global optimum exists and remove constrains of constant speed.

4.4.1 Limitation of the Shifts.

We assume number of shifts forward or backward is unlimited though this presumption is stronger than it is actually needed. The distance between the *first* and the *last* airplane is not greater than $D_{min}(n) = (n-2)(2 \cdot sz + ms) + sz$. Thus aggregated number of steps in either way is limited for every airplane and is equal to $\frac{D_{min}(n)}{ms}$.

To determine the minimal distance from the starting position to the landing point for each airplane only overall maximal time shift of an airplane is necessary. This maximal time shift is equal to $D_{min}(n)$. Denote v_{orig} as original, v_{max} as maximal, v_{min} as minimal speed of the airplane. Suppose $v_{min} < v_{orig} < v_{max}$ and ignore acceleration and deceleration. Then minimal time distance from the landing point to have enough time to shift forward (speed up) is $t_{fwd}(n) = \frac{v_{max}D_{min}(n)}{v_{max}-v_{orig}}$ and to shift backward (slow down) is $t_{back}(n) = \frac{v_{min}D_{min}(n)}{v_{orig}-v_{min}}$. Therefore minimal time distance to have enough time to avoid collisions is $t_{min}(n) = \max(t_{fwd}(n), t_{back}(n))$ and corresponding space distance is $d_{min}(n) = t_{min}(n) \cdot v_{orig}$.

4.4.2 Using Global Optimum.

Another presumption was that the utility function was defined only as local optimization. This condition is used in lemma 1, to show desired selection of the manoeuvres. We show that utility function can also select solutions with regard to global optimal value. With the new utility function lemma 1 is not valid any more. The difference is shown at the Figure 6. In the original definition dashed green airplane (on the left) had no reason to move from its position. But with global optimum placed more at left side, green airplane will shift to the new position (solid circle).

Lemma 1 is used in implication 3 although it can be proved even without it. Using lemma 2 we know, that D has to be increased or stay the same in which case the whole interval on time

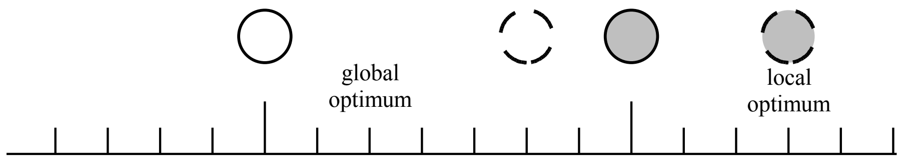


Figure 6: Figure shows different selection of the manoeuvres considering local and global optimum.

axis containing all airplanes have to be shifted after $s(n, D)$ steps. Shift of the whole can occur only when the *first* (*last*) airplane has its global optimum behind (in front of) its actual position. Thus actual interval has to contain global optimum of at least one airplane. Denote D_{orig} as the longest distance between any two global optimums (usually corresponding to starting positions). Size of the time interval, where all airplanes has to be at any time, is $2D_{min} + D_{orig}$ (airplanes can be shifted to the side no more than D_{min} times). As we already proved cycle cannot exist and therefore position of the interval containing all airplanes on the time axis cannot repeat. Thus number of possible positions for the interval is finite and convergence of the algorithm holds.

4.4.3 Different speeds.

We suppose all airplanes have the same original speed and they do not changed it. Same speed is needed to guarantee same size of (time) safety zone as a necessary condition to safely land. What will change, if we permit different speed at the starting points, any where during the flight and disregard scenario with the landing and think just about flight plans with single intersection?

In this case airplanes can fly through this point with different speeds and therefore all airplanes have to know speeds of each other to determine size of a safety zone for each airplane. Algorithm used to solve a collision of a pair of airplanes remains the same, we just need to take into consideration changing safety zones. This allows us also to introduce airplanes with different sizes of the safety zone and with different minimum and maximum speeds.

We can use different airplanes to flight together and use distributed deconfliction algorithm, but then we need to redefine some variables. Minimum (maximum) speed has to be defined as the lowest (highest) speed among all airplanes. sz is defined as a minimum over all airplanes of time needed to fly with minimum speed over its safety zone. With these changes all computations mentioned above are valid.

5 Empirical Analysis

All experiments has been carried out using the framework for airspace domain simulations AGENTFLY described in main report. We denote original algorithm as the *IPPCA ver1* algorithm and the algorithm extended with tendencies as the *IPPCA ver2*. For the experiment scenarios stated in this section we have prepared necessary configurations and special collectors for gathering necessary properties of the algorithm and provide their comparison.

5.1 Pair Collision Analysis

The pair collision experiment is used to study behavior of the IPPCA algorithm for two airplanes. It is measured type of used manoeuvre types for the collisions under different angles, see the Figure 7.

Both airplanes start at the surface of the sphere and follow straight lines across the center to the other side of the sphere. The nominal flight trajectories cross each other in the center. The first airplane (represented by the big red dot) flies from the constant starting position at the sphere. The second airplane starts from positions generated in 5-degree interval. In the horizontal direction is the range from 0 to 360 degrees (72 different positions), in the vertical direction is the range from -20 to +20 degrees (7 different positions). Each configuration is run 10 times. The utility value is optimizing length of the trajectories.

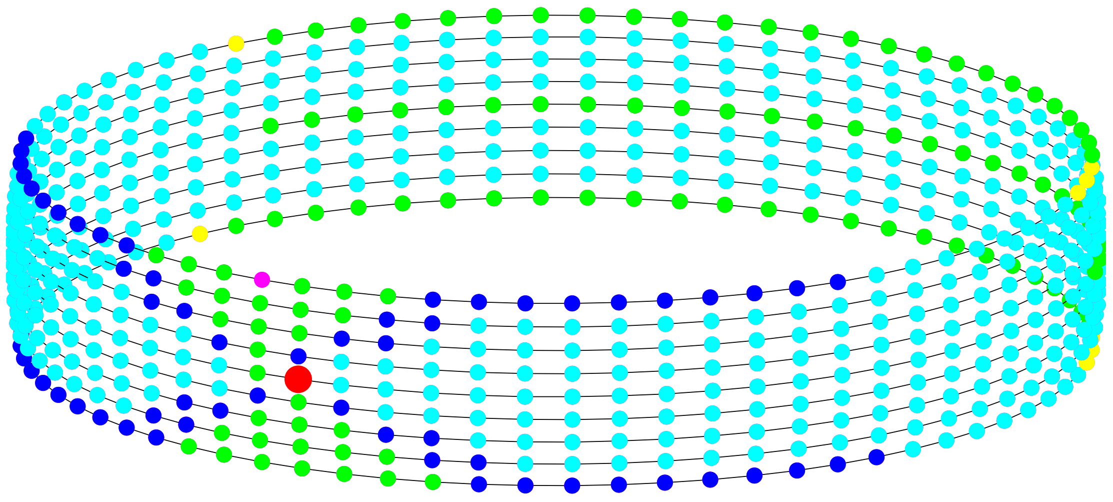


Figure 7: Pairs scenario. Red dot represents direction of the first airplane. The other dots represents the direction of the second airplane. The color specifies type of used manoeuvres.

The manoeuvres are divided into three groups – turn left and right manoeuvres as the *horizontal* manoeuvres, ascend and descend manoeuvres as the *vertical* manoeuvres and speed up and slow down manoeuvres as the *speed* manoeuvres. The colors depicted in the Figure 7 shows combinations of the manoeuvre type for the both airplanes in the following way:

- cyan - vertical + vertical manoeuvres
- green - horizontal + horizontal manoeuvres
- blue - velocity + velocity manoeuvres
- yellow - horizontal + vertical manoeuvres
- magenta - horizontal + velocity manoeuvres

5.2 Landing Scenario Benchmarks

The estimations given by theoretical work described in the Section 4 have been tested in a *landing scenario* with a high density of airplanes in a limited area described in the section 4.1.

In the simulated scenario all airplanes start at the same distance from the landing point and thus each airplane has possible collision with all others. The parameters of the simulated airplanes are: the safety zone 0.5 nmi and the cruise speed 500 knots. The distance from the landing point is taken from the estimation given in subsection 4.4.1 for each experiment. The experiment is tested for 2 to 60 airplanes, 20 repeats per each run.

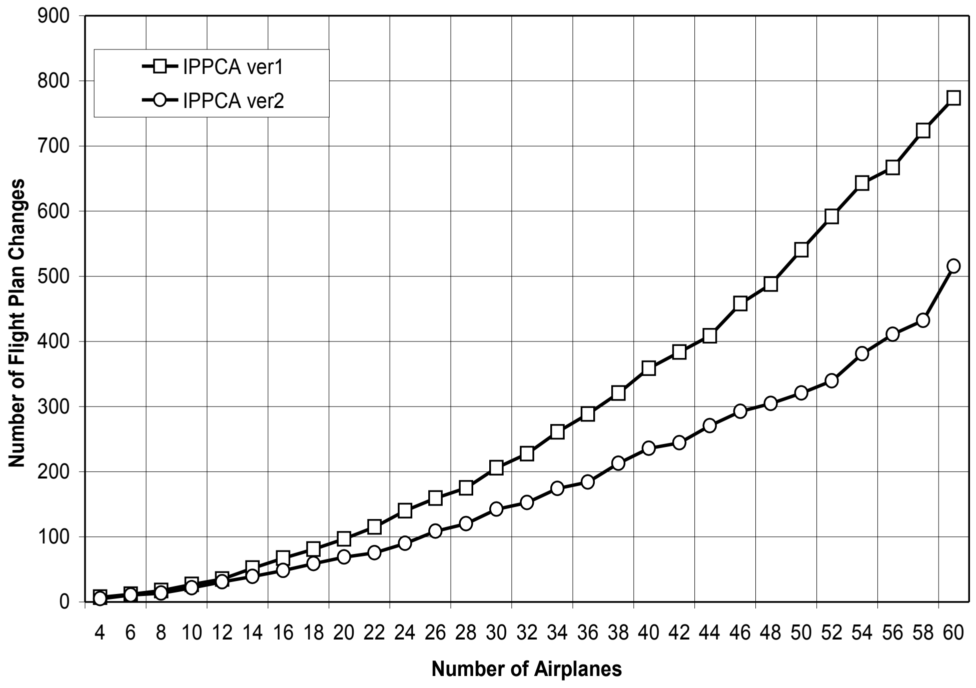


Figure 8: Landing scenario. Number of applied flight plan changes over all airplanes.

None of the experiments records any collision or violation of the minimal separation. Charts in the Figures 8 and 9 show comparison of both IPPCA version. In the Chart 9, we can see the theoretical estimation and optimum of the maximal time difference between the first and the last airplane in the worst case. The *IPPCA ver1* follows linearity of the estimation and is closer to the optimum difference. The *IPPCA ver2* reaches slightly worse results in the time difference needed to solve the problem, but has markedly better results in the number of iterations (number of the applied flight plan changes).

5.3 Perpendicular Flows

The perpendicular flows experiment and circles experiment 5.4 compare results of the IPPCA algorithm to Hill's algorithm 2.1. The configuration is adjusted to follow Hill's description. The IPPCA algorithm is limited to use only turn left and turn right manoeuvres without any change of the speed or altitude.

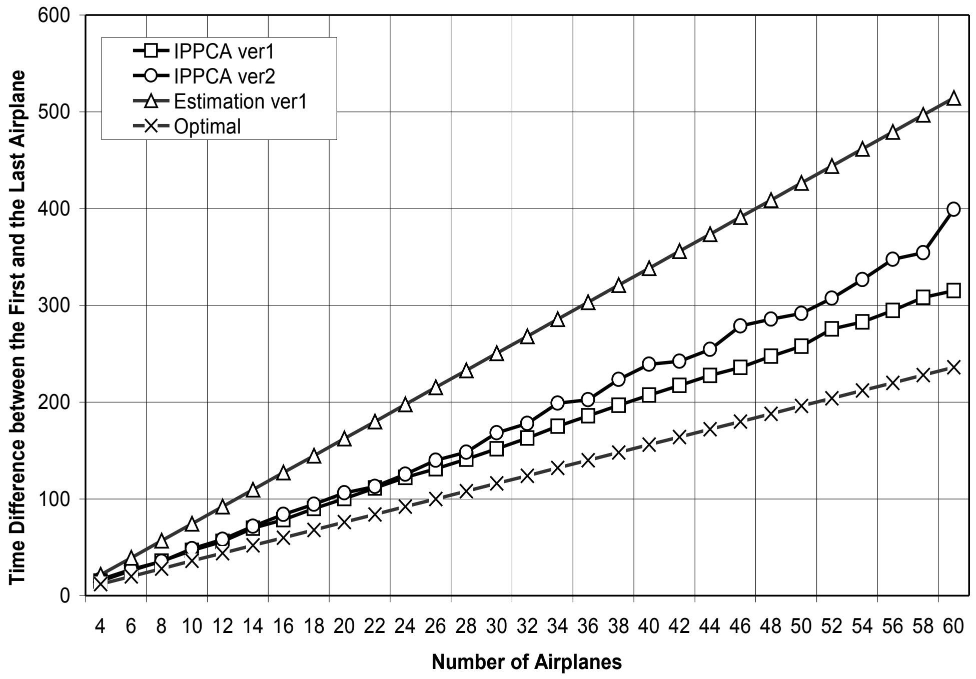


Figure 9: Landing scenario. Time difference between time of arrival of the first and the last airplane.

Near misses (violation of the minimal separation) and system efficiency is measured in all experiments. System efficiency measures the degree to which the airplane follow its nominal flight path [6]. In following experiments the airplanes have linear flight paths and they fly at the same constant speed. Thus the system efficiency can be computed as the rate of the time needed to fly over nominal and real trajectory.

Perpendicular flows scenario is made by two perpendicular linear flows of airplanes. Each Airplane flies at constant speed 500 knots and keeps to avoid violation of the 5 nmi separation distance. Airplanes are aware of all other airplanes within 100 nmi. Airplanes are generated each 40 seconds which preserves about 5.5 nmi distance between each pair of generated airplanes to allow heading changes without immediate violation of the safety zones, see The Figure 10.

The stream of airplanes is interrupted by inserted gaps. A size of the gap is 80 seconds, this means that one airplane is missing in the stream. The average number of airplanes (μ) in the consecutive *string* without a gap is shown in table 1. At Hill's experiments, the string length is a uniform random variable with distribution $U(\frac{\mu}{2}, \frac{3\mu}{2})$. We use *IPPCA ver1* in this scenario and we regularly replace each μ^{th} airplane by the gap. The results are averaged over ten different simulation runs, each modeling a 24 hour period.

Table 1 reports results of the perpendicular flow experiment - corresponding number of flights in Hill and IPPCA period and comparison of the number of the near misses and the system efficiency. We can see that IPPCA algorithm reaches better results in number of the near misses while it preserves slightly better system efficiency. This results are given by possibility of the IPPCA algorithm to apply bigger heading changes than 5 degrees.

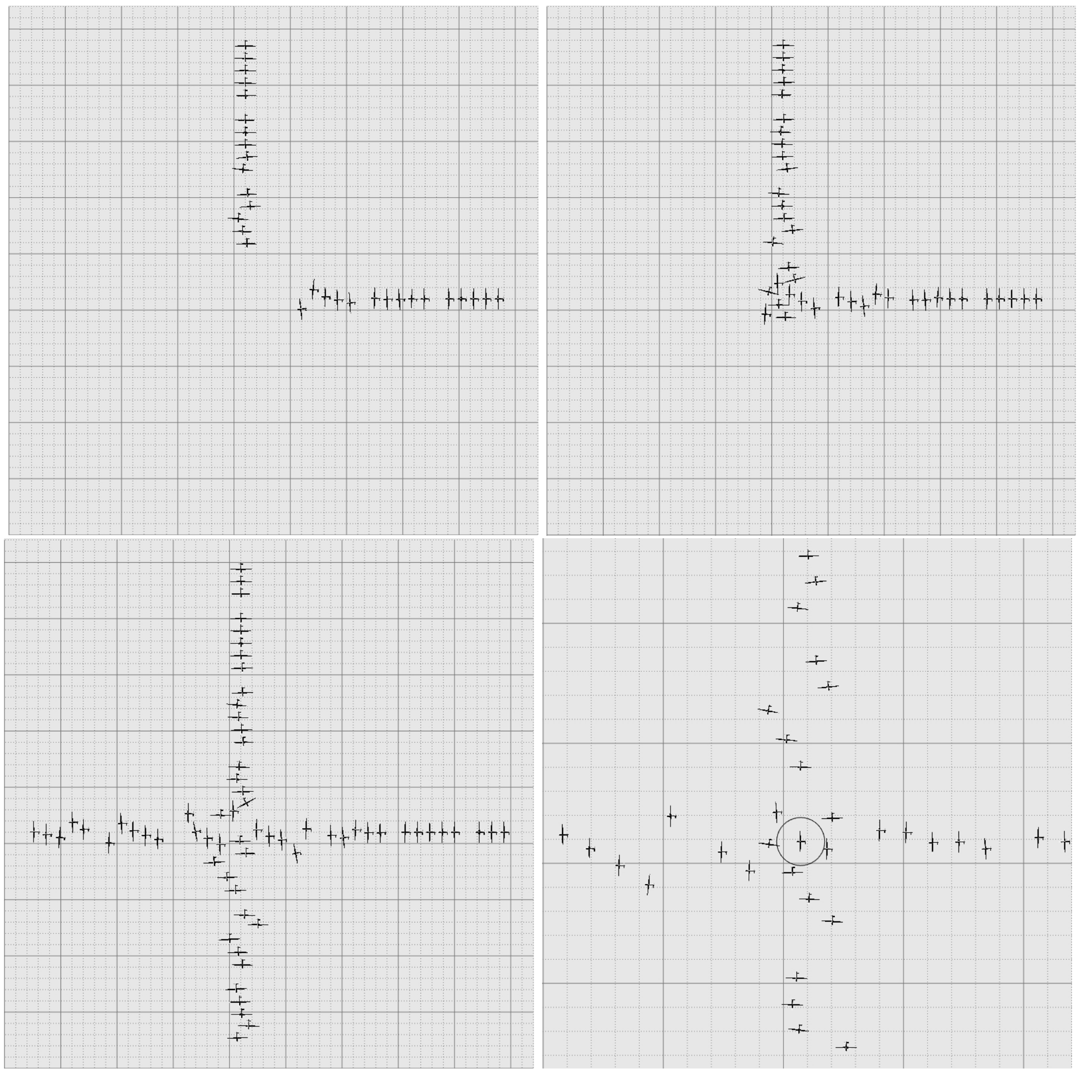


Figure 10: Flows scenario. Simulation for a gap after each 5 airplanes.

Dist.	Flights		Near misses		Efficiency %	
(μ)	Hill	IPPCA	Hill	IPPCA	Hill	IPPCA
4	2732	3240	61	0.1	99.4	99.5
8	3366	3780	378	2.8	98.1	98.7
12	3629	3960	626	5.75	97.1	98.0
16	3767	4050	796	9.6	96.4	97.9
20	3860	4104	1011	11.9	95.4	97.6
const.	4217	4320	1847	31.2	92.3	96.0

Table 1: Comparison of the Hill's and IPPCA algorithm using perpendicular flow scenario.

5.4 Circles

The circle scenario experiment compares the IPPCA algorithm to the Hill's algorithm as described in the section 5.3.

Airplanes in the circle scenario are composed in circle with radius 50 nmi. Each airplane's destination is at the circle exactly opposite to the starting point, thus the nominal flight path coincide at the center of the circle, see the Figure 11. The parameters of airplanes remain same as in the previous scenario. The experiment is tested for 2 to 32 airplanes, 20 repeats per each run.

We measure both variation of the IPPCA algorithm in this scenario. The charts in the Figures 12 and 13 show better results in number of near misses but also worse results in the system efficiency. The number of near misses for 32 airplanes does not follow previous values, because the scenario turns to the scenario with extreme density. These results are given by longer flight paths leading to lower number of separation violations.

5.5 Sphere

The sphere scenario extends the circle scenario described in the section 5.4. It enables full range of manoeuvres including speed and altitude changes. The airplanes are positioned in three layers on the sphere. Middle layer corresponds to the circle scenario. Airplanes at the top layer fly to opposite point of the bottom layer and vice versa. Thus all nominal trajectories intersect in the center of the sphere.

We preserve the minimal separation of the airplanes at 5 nmi and enlarge radius of the sphere to 100 nmi to have enough space for experiments with high density. Nominal slope of the trajectory of the airplanes in top and bottom layer is set to 80% of their maximal ascending/descending angle which is set to 20 degrees. We simulate 20 runs for each experiment from 9 to 60 airplanes (from 3 to 20 in each layer). The simulation is made for both versions of the IPPCA algorithm.

None of the experiments records any collision or violation of the minimal separation. The charts in the Figures 14 and 15 reports the differences between both versions of the IPPCA algorithm. The Figure 14 shows *IPPCA ver2* algorithm needs less iterations to find solution. The Figure 15 shows better efficiency of the *IPPCA ver1*. The system efficiency in this scenario is measured as a rate of a length of the nominal trajectory to a length of the real trajectory.

A Monotonic concession protocol

(MCP) [13, 8] is a classical and widely used protocol for one-to-one negotiation. This protocol organizes negotiation among two actors who are reaching an agreement that is defined as single element of a negotiation set. Each of the actors may gain a different utility value out of the various elements selected from the negotiation set.

The MCP protocols proceeds in rounds. Let us have two agents A and B engaged in the negotiation process over the proposals σ_A and σ_B, elements from the negotiation set. In each round both agents make simultaneous proposals. In the first round each agent is free to make any proposal. In subsequent rounds, the agents have got two options:

– either A or B makes a concession and propose a new deal σ'_A (σ'_Brespectively) that is preferable to the other agent: $u_B(\sigma_A) < u_B(\sigma'_A)$ or
– refuse to make a concession and stick to the proposal σ_A (σ_B respectively).

Agreement is reached if one agent proposes an agreement that is at least as good for the other agent as their own proposal: $u_A(\sigma_B) \geq u_A(\sigma_A)$ or $u_B(\sigma_A) \geq u_B(\sigma_B)$. In the case where both conditions hold a proposal is selected randomly. Conflict arises when we get to a round where nobody concedes. In this case the conflict deal will be the outcome of the negotiation.

With MCP there are several open questions. Who shall concede, how much, etc. There have been several strategies that complement MCP proposed in the research community. Zeuthen proposed a strategy (referred to as Zeuthen strategy) that provides the following set of rules for concession:

– start by proposing the best possible agreement; then
– concede whenever your willingness to risk conflict is less or equal to your opponent's;
– concede just enough to make your opponent's willingness to risk conflict less than yours.

Willingness to risk is calculated according to the following formula:

$$risk_A^t = \begin{cases} 1, & \text{if } u_A(\sigma_A) = 0 \\ \frac{u_A(\sigma_A) - u_A(\sigma_B)}{u_A(\sigma_A)}, & \text{otherwise} \end{cases}$$

If the opponent concedes too much he may obviously 'waste' some of his utility. Often one has to pay for negotiation rounds as it may be beneficial to agree as soon as possible. Small concession would be hence too inefficient.

If both agents use the Zeuthen strategy, then the final agreement maximizes the Nash product. This has first been observed by John C. Harsanyi in 1956. That is, agent A makes a (minimal) concession if and only if its current proposal does not yield the higher product of utilities. Hence, the Zeuthen Strategy ensures a final agreement that maximizes this product. It follows that the final agreement will be Pareto optimal. Zeuthen strategy makes CMP an anytime algorithm (i.e. if terminated at any time the social welfare is better than before). Unfortunately, the mechanism where both agents use the Zeuthen strategy is not stable. MCP has been recently extended for covering multilateral process of reaching an agreement [2].

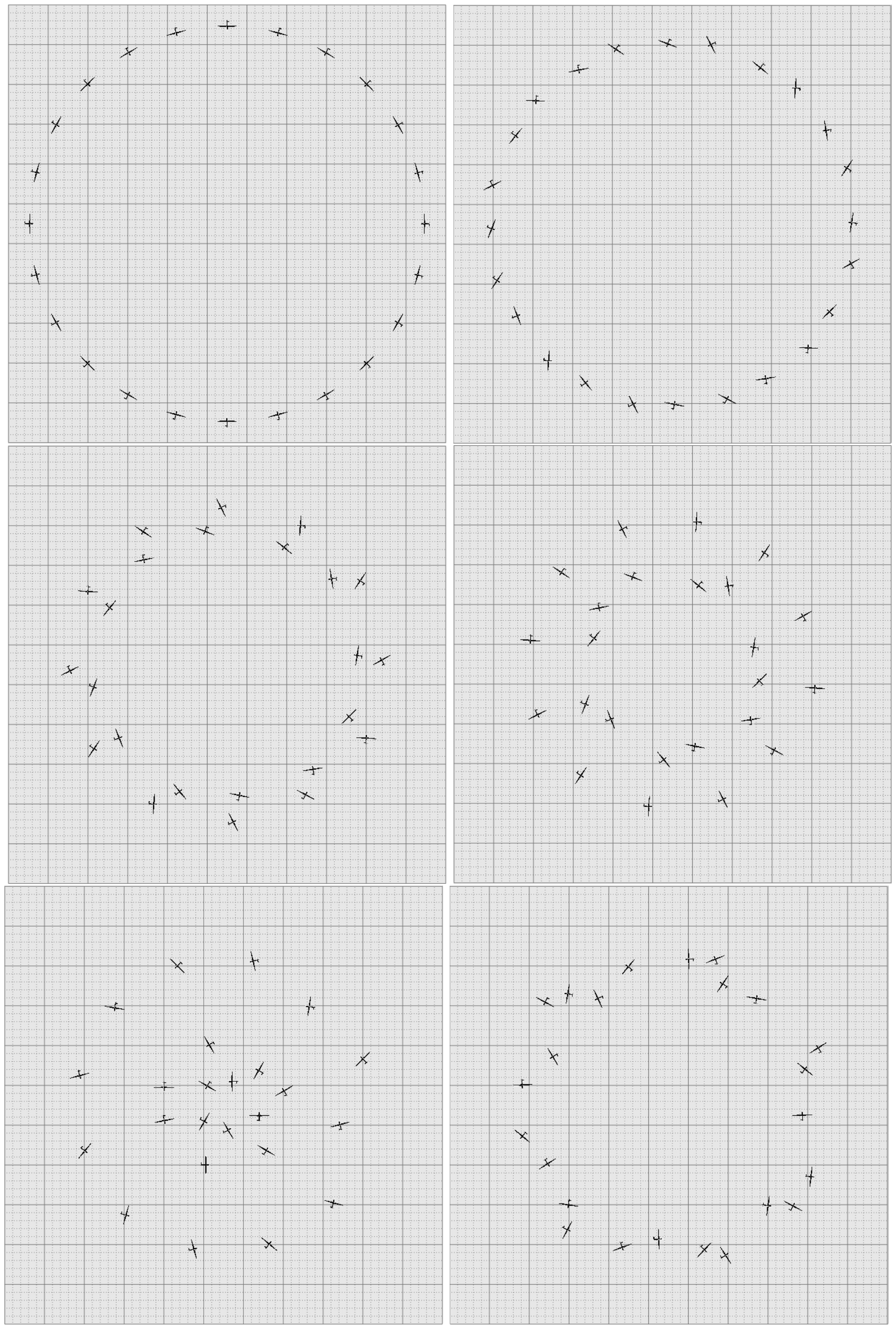


Figure 11: Circle scenario. Simulation for 24 airplanes.

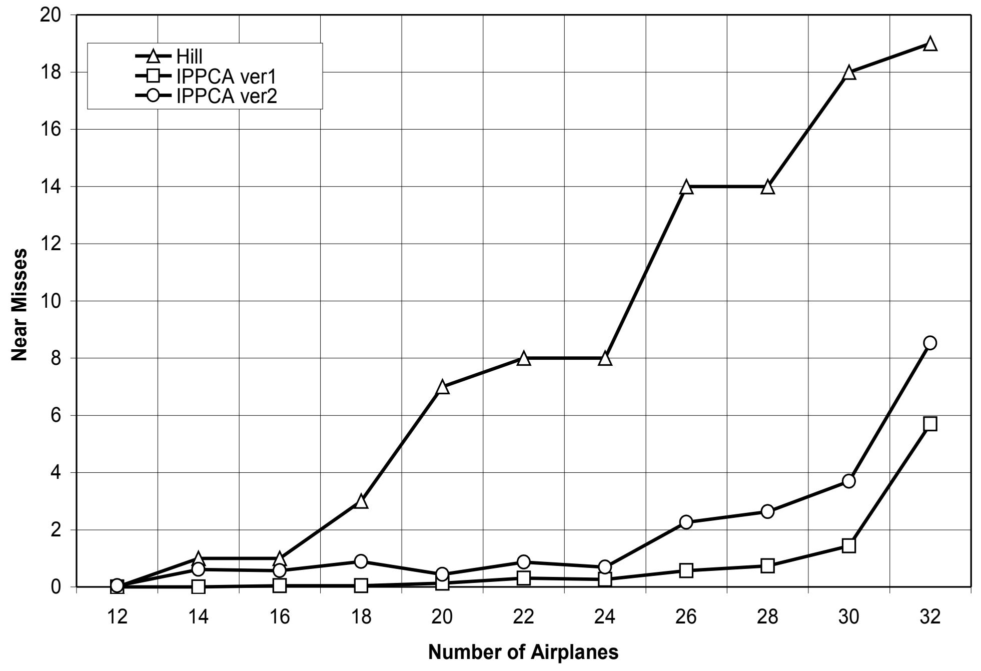


Figure 12: Circle scenario. Total number of minimal separation violation.

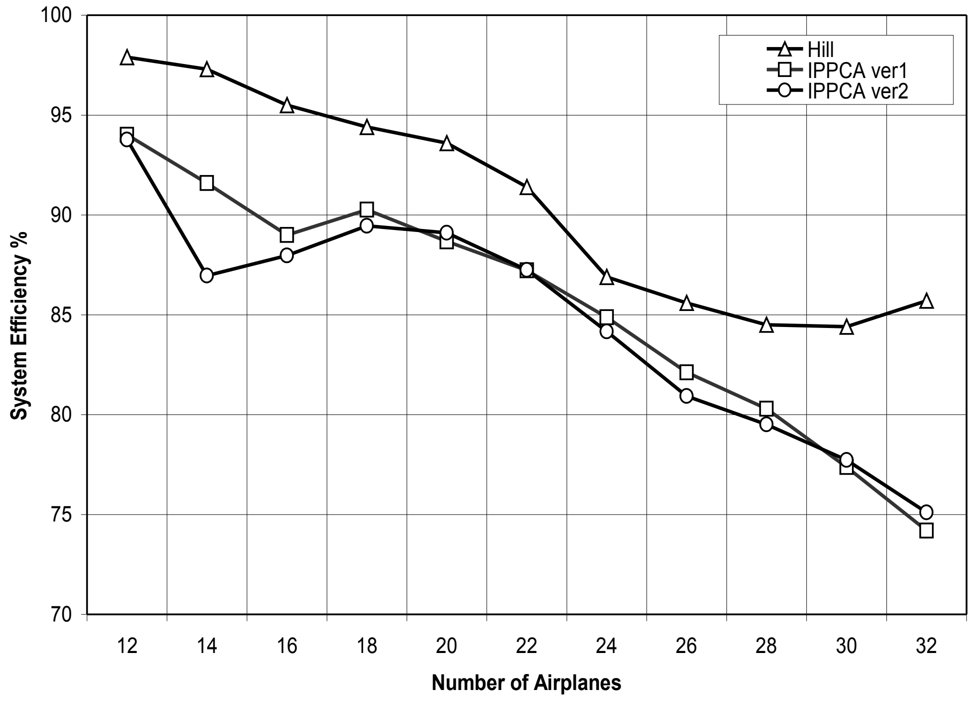


Figure 13: Circle scenario. System efficiency (as a time needed for the flight).

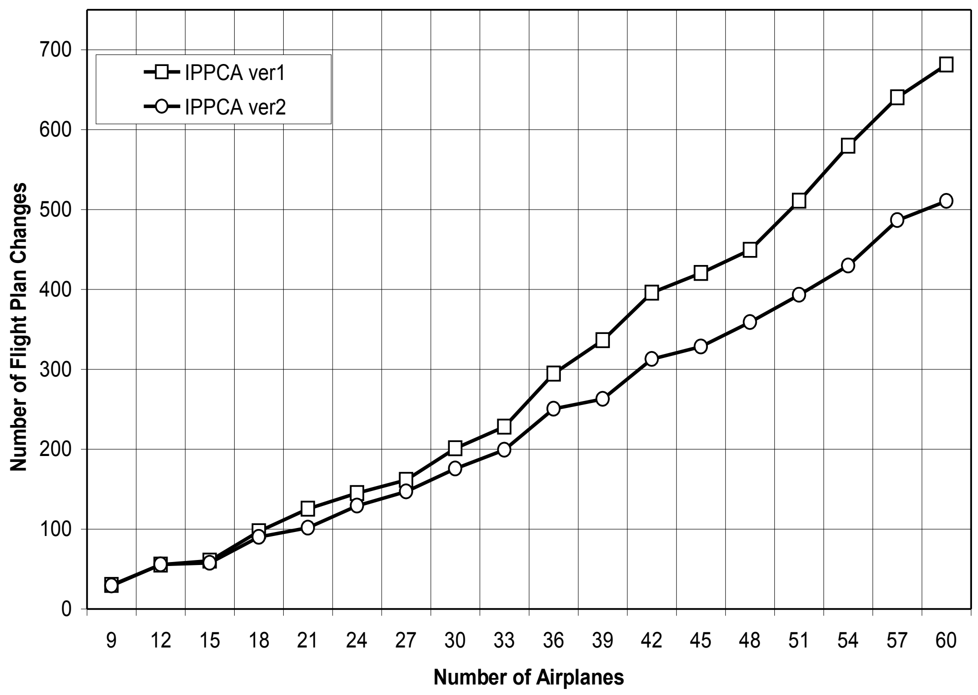


Figure 14: Sphere scenario. Number of applied flight plan changes over all airplanes.

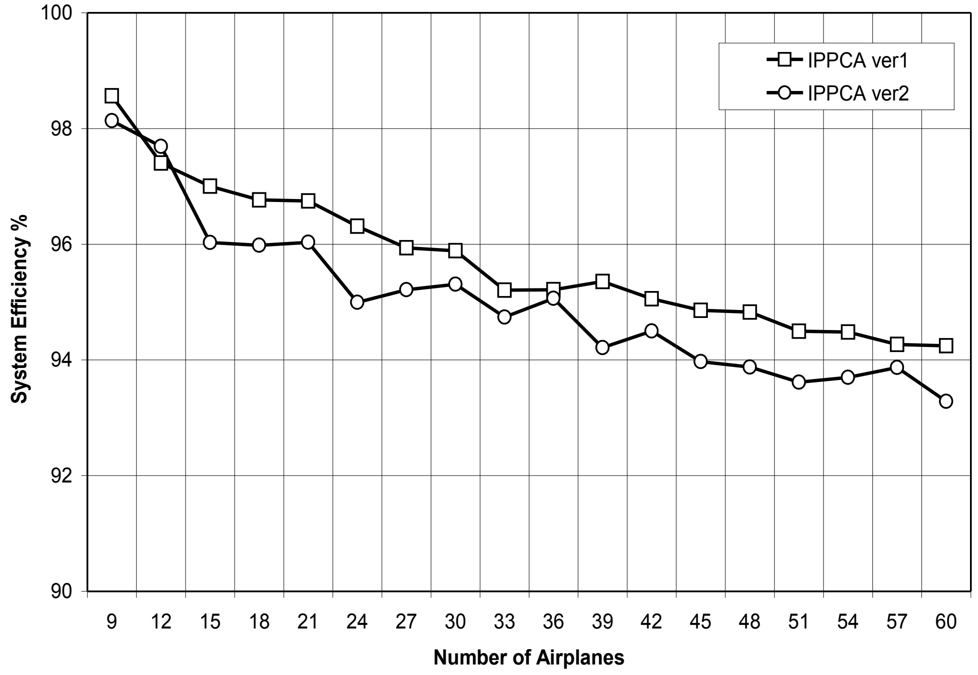


Figure 15: Sphere scenario. System efficiency (as a length of the trajectory).

References

[1] Adrian Agogino and Kagan Tumer. Evolving distributed agents for managing air traffic. In *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1888–1895, New York, NY, USA, 2007. ACM Press.

[2] Ulle Endriss. Monotonic concession protocols for multilateral negotiation. In *AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 392–399, New York, NY, USA, 2006. ACM Press.

[3] Su-Cheol Han and Hyochong Bang. Proportional navigation-based optimal collision avoidance for uavs. In S. C. Mukhopadhyay and G. Sen Gupta, editors, *Second International Conference on Autonomous Robots and Agents*, pages 76–81. Massey University, New Zealand, 2004.

[4] R Holdsworth. *Autonomous In-Flight Path Planning to replace pure Collision Avoidance for Free Flight Aircraft using Automatic Dependent Surveillance Broadcast.* PhD thesis, Swinburne University, Melbourne, Australia, November 2003.

[5] J. Kosecka, C Tomlin, G Pappas, and S Sastry. Generation of conflict resolution manoeuvres for air traffic management. In *Proceedings of Intelligent Robots and Systems*, volume 3, pages 1598–1603, September 1997.

[6] Jimmy Krozel, Mark Peters, Karl D.Bilimoria, Changkil Lee, and Joseph S.B. Mitchel. System performance characteristics of centralized and decentralized air traffic separation strategies. In *4th USA/Europe Air Traffic Management R & D Seminar*, Stanta Fe, NM, December 2001.

[7] Přemysl Volf, David Šišlák, Michal Pěchouček, and Magdalena Prokopová. Convergence of peer-to-peer collision avoidance among unmanned aerial vehicles. In *IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT)*. IEEE, 2007.

[8] Jeffrey S. Rosenschein and Gilad Zlotkin. *Rules of Encounter.* The MIT Press, Cambridge, Massachusetts, 1994.

[9] R Schulz, D. Shaner, and Y Zhao. Free-flight concept. In *Proceedings of the AiAA Guidance, Navigation and Control Conference*, pages 999–903, New Orelans, LA, 1997.

[10] C. Tomlin, G. Pappas, and S. Sastry. Noncooperative conflict resolution. In *Proceedings IEEE Conference on Decision and Control*, December 1997.

[11] John P. Wangermann and Robert F. Stengel. Optimization and coordination of multiagent systems using principled negotiation. *Journal of Guidance, Control, and Dynamics*, 22(1):43–50, 1999.

[12] Steven Wollkind, John Valasek, and Thomas R. Ioerger. Automated conflict resolution for air traffic management using cooperative multiagent negotiation. In *Proc. of the American Inst. of Aeronautics and Astronautics Conference on Guidance, Navigation, and Control*, Providence, RI, 2004.

[13] Gilad Zlotkin and Jeffrey S. Rosenschein. Negotiation and task sharing among autonomous agents in cooperative domains. In N. S. Sridharan, editor, *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 912–917, San Mateo, CA, 1989. Morgan Kaufmann.

AGENTFLY: Autonomous Agents in Air-Traffic Control January 2008 Final Report

this is a project final report to the FA8655-06-1-3073 contract and it provides technical information about complete work on the 'Autonomous Agents in Air-Traffic Control' project

Michal Pěchouček[PI], David Šišlák,
Přemysl Volf, Štěpán Kopřiva, Jiří Samek

Agent Technology Group, Gerstner Laboratory,
Czech Technical University in Prague

Contents

1 Executive Summary

This final report of the AFRL project FA8655-06-1-3073 provides the complete description of the AGENTFLY system. The AGENTFLY is the prototype application of the multi-agent system deployed to the airspace domain mainly for the distributed agent-based collision avoidance among autonomous aerial assets (UAA) and for the collective flight modeling. While to core work of the project has been developed with FA8655-06-1-3073 project (referred here as *FA8655-06-1-3073-main*), smaller part of the work was supported by a specific project extension (referred here as *FA8655-06-1-3073-extension*). The extension partially supports implementation of the collective flight modeling described in this document. The main part of the FA8655-06-1-3073-extension was aimed at the theoretical study of the properties of iterative peer-to-peer collision avoidance, identification of the worst case scenarios and scalability tests providing the empirical properties of the used technology compared to the latest state-of-the-art. The results from these topics are stated in the separate project extension report [9].

The key implemented workpieces within FA8655-06-1-3073-main are described in this report that is structured into several sections providing:

- the use modes of the AGENTFLY prototype (Section 3),
- the detailed specification of the airspace domain where the multi-agent techniques have been applied (Section 4),
- the flight modeling (Section 5) and time-constrained way-point flight plan planning algorithm in defined airspace avoiding the ground surface and no-flight zones (Section 6),
- the description of the AGENTFLY prototype architecture (Sections 7 and 9) including real-time visualization component (Section 11), remote web-based access (Section 10) and the integration with various external data sources (Section 8),
- the multiple operator agent interface providing the human-system interface allowing the real-time control of the UAAs (Section 12),
- the multi-layer collision avoidance architecture allowing simultaneous combination of the cooperative and non-cooperative collision avoidance methods (Section 13),
- the description of three implemented cooperative collision avoidance algorithms (Section 14): *rule-based*, *iterative peer-to-peer* and *multi-party collision avoidance*,
- the non-cooperative collision avoidance based on the dynamic no-flight zones (Section 15),
- the collective flight coordination architecture used for the synchronization of the group of UAAs supporting the flight in the formation as well as the outer group-to-group negotiations (Section 16),
- the basic set of the empirical experiments (Section 17) comparing all cooperative methods together, non-cooperative no-flight zone method comparison to the optimal proportional navigation algorithm,
- the specialized demo cases demonstrating the benefits of the multi-party algorithm (Section 17.1.3) and validating the concept of multi-layer collision avoidance architecture where the agent controlled airplanes operate in the area with civil traffic (Section 18),
- the description of the complex combat scenario used as a testing case for the mix of all features provided by AGENTFLY in a very complex mission (Section 19),
- the AGENTFLY prototype runtime requirements (Section 20).

2 Publications

The work from this project is described in the several already published (or accepted) publications:

[1] David Šišlák, Jiří Samek, and Michal Pěchouček. Decentralized algorithms for collision avoidance in airspace. In Padgham, Parkes, Mueller, and Parsons, editors, *Proceedings of 7th International Converence on Autonomous Agents and Multi-Agent Systems (AAMAS 2008)*, to appear.

[2] Davis Šišlák, Michal Pěchouček, Přemysl Volf, Dušan Pavlíček, Jiří Samek, Vladimír Mařík, and Paul Losiewicz. *AGENTFLY: Towards Multi-Agent Technology in Free Flight Air Traffic Control*, Defense Industry Applications of Autonomous Agents and Multi-Agent Systems, chapter 7, pages 73–97. Birkhauser Verlag, 2008.

[3] Jiří Samek, David Šišlák, Přemysl Volf and Michal Pěchouček. Multi-party collision avoidance among unmanned aerial vehicles. In *2007 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT 2007)*, pages 403–406, November 2007.

[4] Vladimír Mařík, Libor Přeučil, Michal Pěchouček, Miloslav Kulich, Milan Rollo, Roman Mázl, Pavel Vrba, Tomáš Krajník, and David Šišlák. *Od osamocených robotů ke kolaborativní robotice*, Umělá inteligence (5), pages 431–496. Academia, Praha, 2007.

[5] Pavel Vrba, Vladimír Mařík, Libor Přeučil, Miroslav Kulich, and David Šišlák. Collision avoidance algorithms: Multi-agent approach. In *Lecture Notes in Computer Science of Holonic and Multi-Agent Systems for Manufacturing - HoloMAS 2007*, pages 348–360, Munich, 2007. Springer.

[6] David Šišlák, Přemysl Volf, Antonín Komenda, Jiří Samek, and Michal Pěchouček. Agent-based multi-layer collision avoidance to unmanned aerial vehicles. In *Proceedings of International Conference on Integration of Knowledge Intensive Multi-Agent Systems (KIMAS '07): Modeling, Evolution and Engineering*, 2007.

[7] Michal Pěchouček, David Šišlák, Dušan Pavlíček, and Miroslav Uller. Autonomous agents for air-traffic deconfliction. In Peter Stone and Gerhard Weiss, editors, *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 1498–1505. ACM, 2006.

[8] David Šišlák, Martin Rehák, Michal Pěchouček, Dušan Pavlíček, and Miroslav Uller. Negotiation-based approach to unmanned aerial vehicles. In *DIS '06: Proceedings of the IEEE Workshop on Distributed Intelligent Systems: Collective Intelligence and Its Applications (DIS'06)*, pages 279–284, Washington, DC, USA, 2006. IEEE Computer Society.

3 AGENTFLY Usage Modes

The designed AGENTFLY *collision avoidance* system described in this document can be used in several modes. The main goal considered during the system development was to use the multi-agent system for **real-time planning and control**. The multi-agent system integrates the concept of *Free Flight* and thus the initial flight path planning by airplanes (when they are preparing to start or their mission is amended) provides the plans that can contain possible collisions. This makes initial planning very fast fulfilling only constraints given by the airplane model and airspace (ground and defined no-flight zones). The collisions are then detected and solved by *See and Avoid* capability of *Collision Avoidance* algorithms just during their flight execution. The setup of detection range defines how much in advance the collisions are removed and thus restricts the number of airplanes which detects mutual collisions. The avoidance algorithm then solves the conflicts of that identified groups of the airplanes. The reduction of the number of airplanes leads to the reduction of the algorithm complexity. Airplanes can fly on any planned flight corridor not just on predefined ones. This results in substantial scalability improvement affording higher number of aerial vehicles to operate in condensed airspaces.

Described real-time case can be realized in three different deployment setups: *fully deployed* case, *remote control* case and *hybrid deployment*. In the first setup the agents providing described function are running on the asset hardware and use on board equipment. The agents don't need to communicate with any headquarter center (or airspace control center) to provide collision-free corridors. In the *remote control* case the agents are running on a server farm in control center (ground or air-commandn center like EC130J) and each *plane agent* (Section 9.2.1) has permanent communication link to control respective asset remotely. In this case it is not necessary to have whole multi-agent system on the same server farm. The agents (and thus UAAs) can be split into several groups running on own host but to provide cooperative collision avoidance (Section 14) there must exists a communication connection among them. If the cooperative collision avoidance between any asset from different group is not necessary, the airplanes use non-cooperative avoidance (Section 15) which requires radar sensing function. It is allowed to read radar data from the asset hardware via communication link or use control center radar sensing. The multi-agent technology allows simple combination of described setups in *hybrid deployment* – some agents are running on the assets and others control them remotely using on-board or off-board radars still using agent-to-agent negotiations where it is possible.

The AGENTFLY system can be used also in **off-line planning** mode. In this mode the system utilizes its capability of detailed flight modeling (Section 5). The system provides simulation and collects the results: detailed collision free flight corridors, contacts times etc. In this mode the system virtual time is simulated in as fast as possible manner – the simulation speed is automatically adjusted depending on the system load. The time necessary for getting the results depends on the complexity of the given situation and available resources. The use of the system in off-line planning mode has sense only in the situations when we know all flight details of all aircrafts in advance and is not suitable for the cases with at-least one non-cooperative (not controllable, intruder or enemy) airplane.

The simulation capability can be used also for the **verification** of given corridors and control requests. In this mode agents don't provide any new control for airplanes and all collision avoidance algorithm are disabled. They just follow the given instruction and the simulation environment checks if all specified constraints are satisfied.

Beside the collision avoidance the multi-agent system provides also *collective flight coordination* (Section 16), *human on-line interactions* (Section 12) and is open to integrate any other advanced control algorithm (both distributed and centralized). The collective flight coordination can operate in all described modes but human on-line interactions as well as non-cooperative collision avoidance can be utilized only in the real-time mode.

4 AGENTFLY Domain Description

In this section the detailed specification of the free-flight domain (FFD) as provided by AGENTFLY is described. In FFD the autonomous aircrafts, all members of $\mathcal{A}$, operate in a shared three-dimensional *airspace* $Air \subseteq R^3$ that is limited by the ground surface and airspace boundaries, see the Section 4.1. The behavior of the airplane is given by its flight plan (see Section 4.2). The Section 4.3 introduces several zones around each airplane. Collisions and multi-collision sets are described in the Section 4.4. The collision avoidance problem is defined in the Section 4.5. The use of the local collision avoidance is essential in the cases with restricted communication as described in the Section 4.6.

4.1 Airspace

The airspace Air_i that can be occupied by an individual aircraft A_i is made smaller by *no-flight zones* $\mathcal{Z}_i = \{Z_1, Z_2 \ldots\}$, where the A_i cannot fly (thus $Air_i = Air - \sum_{Z_k \in \mathcal{Z}_i} Z_k$). The AGENTFLY system internally represents airspace as a block subsection of R^3 defining the boundaries and a set of the no-flight zones. For the simpler implementation the ground surface is defined as a special case of no-flight zone. Thus the AGENTFLY divides no-flight zones into three groups:

- *world zones* – represent ground terrain and other static obstacles in the simulated world,
- *static zones* – encapsulates world areas where UAA cannot operate: strategic places (e.g. nuclear power plants), non-standard weather conditions or dangerous enemy territories,
- *dynamic zones* – holds zones which are changed very often; they are mainly defined by non-cooperative collision avoidance algorithm (Section 15.1.2).

The basic building blocks for defining no-flight zones are:

- *octant tree* – the area described by the tree is divided by eight identical cuboidal subcells [3]. All cells which are fully occupied by that zone are marked as a full cell. Cells which are not intersected by zone are marked as the empty ones. There are many cells marked as mixed which have defined another eight subcells with own status. The maximum height of the tree structure can be limited to provide fast test operations.
- *height map* – the internal matrix structure defines how many space is occupied over position identified by respective cell indexes. The matrix is initially filled from the height map images where the intensity of the pixel defines the height (brighter is higher and darker is lower). The implementation allows also to specify neighbors aggregation factor doing reduction of the matrix size and thus speed up testing operations on it.
- *add/sub composition* – defines the no-flight regions by adding and substraction operation on elementary geometrical objects: block, ellipsoid and cylinder. Using such composition very complicated zone shapes can be build. This representation allows not only fast line intersection testing but also fast corridor intersection testing.

The final zone shape is defined as a join group operation of several basic building blocks. It looks like hierarchical tree structure where each leaf node in the tree is one of the basic block and internal nodes are grouping and/or transformation operations. The root node of the tree integrates the basic boundaries of whole airspace as a block subsection of R^3. Thus the area defined by any basic block can be simply rotated, scaled and translated as defined in the node transformation:

$$\overline{x}' = \mathbb{R} \cdot \overline{x} - t,$$

where $\mathbb{R}$ is rotation matrix t is translation vector, $\overline{x}$ is point in node system of coordinates and $\overline{x}'$ is point in the system of coordinates below this node.

The hierarchical structure defining airspace Air_i for the airplane A_i provides interfaces to perform the intersection testing for point, line and also corridor with given dimensions. For line and corridor it is able to return the first intersection point on that path. The testing of corridor on octant tree and height map basic block leads to several consecutive calls of transformed line tests on that structures.

4.2 Flight Plan

The behavior of the airplane $A_i \in \mathcal{A}$ is given by the specific *flight plan* $fp_i = (sp, e_1, e_2 \ldots e_n)$ defined as a start situation and ordered sequence of flight elements. The airplane state at time t including position, direction vector, normal vector and velocity is defined as $\sigma(t) = \langle \overline{x}, \overline{d}, \overline{n}, v \rangle$ where t is counted from the start of the fp_i. The start situation includes the initial airplane state and start time $sp = \langle \sigma(0), t_0 \rangle$. Each of the flight elements e_i can be one of the following types (see Figure 1):

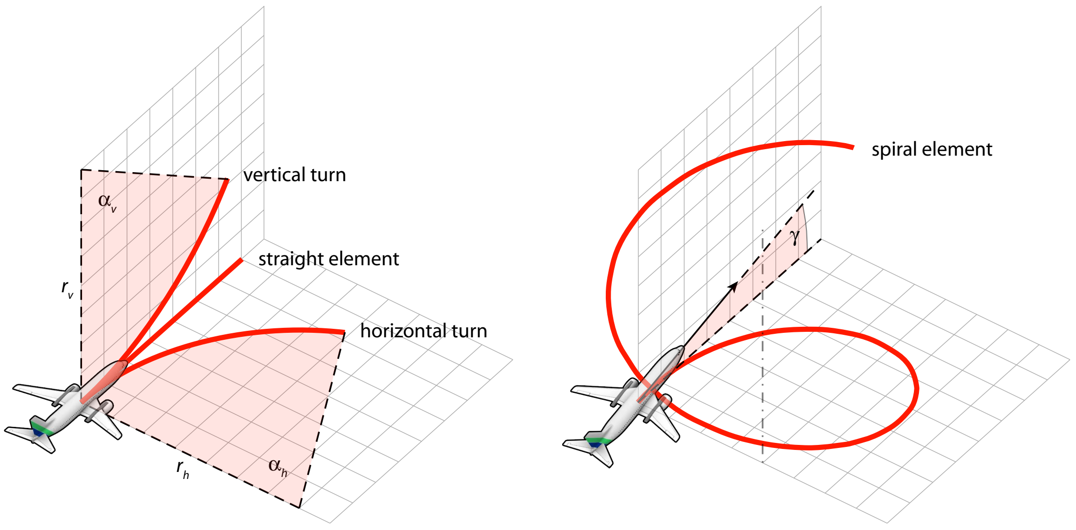


Figure 1: Flight plan element types

- *Straight element* $e_{straight} = \langle l, a \rangle$ – specified by its length and acceleration. The airplane will simply fly with the constant acceleration (the acceleration can be zero) for the specified time in the direction given by its initial state in this element $\sigma_{e_i}(0)$. The final velocity (at the end of the element) is determined by the initial velocity, duration and acceleration.
- *Horizontal or vertical turn elements* $e_{turn} = \langle r, d \rangle$ – given by the turn radius (sign defines turn direction) and angle. They are represented by circular arcs. Horizontal turns are performed

in the plane of flight. Vertical turns are performed in the plane described by the initial airplane direction and normal vectors. The vertical turns are used to represent parts of the flight path, where the plane changes its motion from a horizontal flight to an ascent and descend trajectory.

- *Spiral element* $e_{spiral} = \langle r, d, c \rangle$ – similar to horizontal turn element extended by a climbing rate. It is represented by a part of a spiral with an axis parallel to the z axis of the world coordinate system. The parameters of the spiral are the radius (again the sign defines turning direction), duration and climbing rate (sign define if ascend or descend). The climb rate specifies how the altitude is changed on one turn.

For the simplification of the modeling and possible execution of such prepared elements the velocity can be changed only on the straight elements and on all others it remains constant. The AGENTFLY implementation contains one more element *warp element*. It is a special element which doesn't contain any trajectory definition. It is defined just as a state σ. It is internally used for the reduction of the cumulative errors given by the differential description of the flight trajectory.

The elements are constrained by the airplane type that often specifies minimal and maximal flying velocity, minimal and maximal acceleration, minimal turn radius and max climbing/descending rate. Each element e_{k+1} specifies the flight path part relatively from the end of the previous element e_k or from sp for the first element e_0. Thus the path given by fp_i is continuous and must be smooth. First derivation in all coordinates and time must be continuous too. Let us introduce $\mathcal{FP} = \{fp_i\}_{A_i \in \mathcal{A}}$ as a set of all actual flight plans of the aircrafts in the airspace.

The behavior of an aircraft is not random, while it is specified by a *mission*. All points on the path of fp_i must be in Air_i and must be constrained by the airplane mission $M_i = \langle wp_1, wp_2 \ldots wp_n \rangle$, a sequence of *way-points* $wp_k = \langle \overline{x}, t_1, t_2 \rangle$. The way-point in Air_i specifies the time interval by specifying t_1 and t_2 when the aircraft is allowed/requested to fly through. Let us introduce the function $\mathbf{p}(fp_i, t)$ that returns the position of the individual airplane A_i at the given time t. The way-point wp_k is *completed* by the A_i's flight plan fp_i if $\exists t$ so that $\mathbf{p}(fp_i, t) = \overline{x}^{wp_k}$ and $t_1^{wp_k} \leq t \leq t_2^{wp_k}$. In the other words, the path defined by fp_i must go through each way-point at least once in the specified order of M_i. A segment represents a part of a flight plan between two successive way-points. The example of the flight plan is in the Figure 2.

Definition 1. The **planning problem** in FFD from the perspective of an individual aircraft A_i with respect to the mission M_i is the process of finding such a flight plan fp_i so that $\forall wp_k \in M_i$ are completed.

Airplane can alter its own current fp_i anytime, but only the future part can be changed. The processes of (re)planning and collision avoidance are carried out in the same time as the process of mission accomplishment. Thus, the airplane is allowed to change its flight plan in some future time t to be able to apply new changed fp'_i.

4.3 Zones Around Asset

Each airplane is surrounded by a number of concentric spherical zones, see Figure 3. The AGENTFLY uses the same zone definition for all airplanes of a certain type. Thus, there can exists airplanes with different zones at the same time.

- *Communication zone* – is the outermost one. It represents the communication range of the data transmitter on the aircraft board. See the Section 4.6.

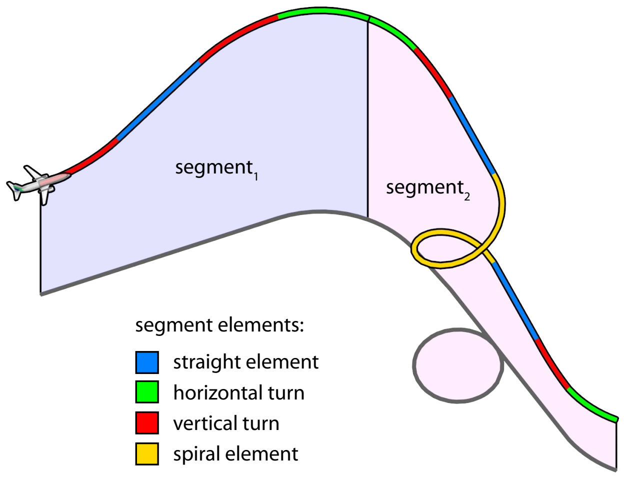


Figure 2: The example of the flight plan

- *Alert zone* – defines the operation range of the radar sensing available to the agents controlling the airplane. If another airplane is located within the zone, agents are periodically notified about its relative position. This situation is shown in the Figure 3. The radar data are used for the non-cooperative collision avoidance (Section 15) and can be also used for the checking known information from the others.
- *Safety zone* – is used for the separation planning of airplanes during collision avoidance. The detailed description is in the Section 4.4.
- *Collision zone* – is the innermost zone. It defines critical contact area. When two airplanes get too close together and their mutual distance is smaller than the sum of their collision radiuses, physical contact between them occurs.

4.4 Collision Definition

Around each airplane there is defined a *safety zone* – a spherical space with a given radius rsz_i for each A_i. It defines surrounding airspace where no other airplane is allowed to appear so that effects of turbulence caused by other airplane and inaccuracy of flight plan execution (there are allowed small deviations from the flight path) can be avoided. Let us introduce the function

$$\mathbf{col}(fp_i, fp_j, t) = \begin{cases} 1 & \text{if } |\mathbf{p}(fp_i,t),\mathbf{p}(fp_j,t)| \le \max(rsz_i, rsz_j) \\ 0 & \text{otherwise} \end{cases}$$

specifying the fact that two flight plans fp_i and fp_j have a *collision* in time t.

Definition 2. Two aircrafts A_i and A_j (with their flight plans fp_i and fp_j respectively) are **colliding** (denoted as $A_i \otimes A_j$) if and only if $\exists t : \mathbf{col}(fp_i, fp_j, t) = 1$.

Clearly, $A_i \otimes A_j \equiv A_j \otimes A_i$. The set $\mathcal{A}$ is dynamic as there can be new planes created or removed (e.g. after landing) during the process of mission execution. In the FFD it is guaranteed that newly created plane A_i has no collision on its flight plan fp_i with any other existing fp_j in

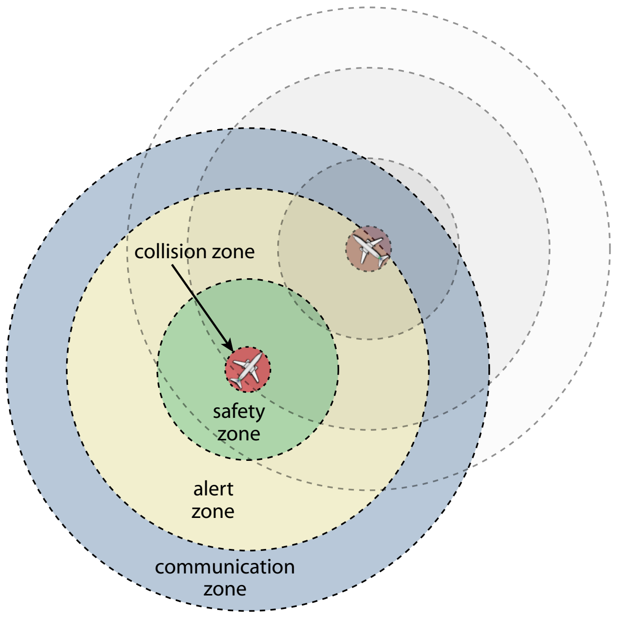


Figure 3: Communication, Alert, Safety and Collision Zones around each aircraft

next δ from its start. The δ value is specified for each aircraft and guarantees that there is enough air space to avoid future collision which appears just after plane creation. After δ the flight plan of the newly created aircraft can collide.

Let us discuss how the multiple collisions can influence each other. We introduce $\mathcal{C}_{all} \subset \mathcal{A} \times \mathcal{A}$ as a set of all colliding aircrafts. We will be working with the *multi-collision set* of collisions $\mathcal{C} \subset \mathcal{C}_{all}$ that includes all related collisions. In $\mathcal{C}$ there is at least one pair of colliding airplanes $A_i \otimes A_j$ and in the same time there is no such collision $A_k \otimes A_l \in \mathcal{C}$ so that neither A_i nor A_j does not have a collision with either A_k or A_l and there is no other collision in $\mathcal{C}$ that is linked with both $A_i \otimes A_j$ and $A_k \otimes A_l$ by a finite number of collisions. Let us view $\mathcal{C}$ as undirected graph. Let us assume that each collision from the set $\mathcal{C}$ has one vertex in a graph, an edge between any two vertices exists if and only if there is at least one A_i involved in both collisions represented by vertices. The $\mathcal{C}$ is the multi-collision set if and only if its graph representation is connected (for every pair of distinct vertices in the graph there exists a path linking them both). Note that the concept of multi-collision set includes also collision of two airplanes only. $\mathcal{A}_{\mathcal{C}} \subseteq \mathcal{A}$ is the set of all aircrafts which are implied in at least one collision in $\mathcal{C}$.

4.5 Collision Avoidance Problem

Let us work with the encounter [11] as a subject of collision avoidance problem. For a given multi-collision set $\mathcal{C}$ an *encounter* en_k is tuple $\langle t, \{fp_i\}_{A_i \in \mathcal{A}_{\mathcal{C}}} \rangle$ such that $t \geq \texttt{now}$ is a time point in the future from which the flight plans of the colliding airplanes can be changed.

Clearly, the *collision avoidance problem* (CAP) in FFD can be defined as the process of finding such $\mathcal{FP}$ for which $\mathcal{C}_{all} = \emptyset$. In this paper we will be solving CAP by solving *local collision avoidance problems* (LCAP) applied on top of $\mathcal{FP}$.

Definition 3. **Local collision avoidance problem** (LCAP) (*replanning*) with respect to an encounter $en_k = \langle t, \{fp_i\}_{A_i \in \mathcal{A}_{\mathcal{C}}} \rangle$ and $\mathcal{FP}$ is the process of finding such a solution $\{fp'_i\}_{A_i \in \mathcal{A}' \subseteq \mathcal{A}}$

founded in time $t' < t$ so that the encounter en_k is eliminated.

The current time and time t from encounter gives the maximal interval in which LCAP algorithm can search for the solution. The selection of right time t in encounter is the part of the algorithm and can take into account its own properties. Two CAP algorithms applied to the same situation can be compared using their final flight plan *utility values* after accomplishment of all aircrafts' missions. The utility function value $\mathbf{u}_i(fp_i)$ used for comparison [7] includes aircraft's intention to proposed solution of the conflicts – e.g. be as short as possible, use minimum of fuel, fulfill time constraints of the way-points, etc. The $\mathbf{u}_i(fp_i) \in \langle 0, 1 \rangle$ is evaluated as weighted sum of its components. Each airplane can have different components, but each airplane must use the same in both compared CAP algorithms. E.g. for the social welfare criterion we can say that one CAP algorithm is better if $\sum_{A_i \in \mathcal{A}} \mathbf{u}_i(fp_i)$ is higher where fp_i represents final flight plan of airplane A_i after applying CAP algorithm.

4.6 Communication Restrictions

The airplane asset can host one or more *agents* and provide them communication infrastructure via its on-board communication transceivers with limited range of communication c_i. So, the agents at A_i can negotiate with agents at A_j in time t only if $\mid \mathbf{p}(fp_i, t), \mathbf{p}(fp_j, t) \mid \leq \min(c_i, c_j)$. The agents on airplane A_i have full information about its flight status and can call functions for altering fp_i. Using this transceiver airplane agents are aware of existence of other airplane if the airplane can communicate with them. There is no central element in the domain so the agent knows only information which can be obtained from its hosting airplane or by negotiation with other agents. Even though the range c_i is relatively large not all aircrafts can communicate together. The domain allows also airplanes which are not capable to communicate with others due to several reasons: broken transceiver, or they don't want to communicate. But in the current system we are focused only on the case where all airplanes $\mathcal{A}$ are able to communicate together if distance condition is satisfied. Thus, agent controlled airplanes can cooperate to do collision avoidance.

5 Flight Simulation

This sections provide the detailed description of the flight simulation in the AGENTFLY. The flight plans are executed by airplanes in discrete fashion with variable time step length, see the Section 5.1. The airplane model and the structure of the flight plans were designed for very fast and scalable simulation of a great number (hundreds to thousands) airplanes at once. That is why we use a simplified physical model of airplanes as described in the Section 5.2. The current implementation doesn't consider the impact of the forces affecting the airplane to the fuel consumption, the decrease of the airplane weight due to the gradual fuel consumption and the influence of wind force on the airplane flight. Moreover the flight velocity can be changed only on the straight flight elements (see Section 4.2). The AGENTFLY currently uses the physical model for the standard monoplane and others such as choppers are not supported in the current AGENTFLY implementation, but the system can be extended by specialized flight models and path planners.

5.1 Flight Execution

The simulation of the flight of airplanes is performed by the plane simulation agent, which resides on a server container (see the AGENTFLY architecture in the Section 7). This agent maintains the information about the state of all airplanes present in the system, along with their flight plans. The flight of airplanes is simulated by the evaluation of their flight plans over the time using flight model described in the Section 5.2. The simulation agent is able to simulate the flight of many airplanes at once. The flight paths of individual airplanes are not fixed after their first planning. They can be changed in future (for example, if two flight plans were set on a collision course, the airplanes will change their flight plans to avoid the collision as described in the Section 4.2.

The simulation agent uses the notion of *global simulation time*. It is the time of a global clock, called simulation clock, running on the simulation agent. This time serves as a reference frame for all time information stored in the flight plans. When a new airplane is created, a free "plane slot" is allocated in the plane simulator for its flight plan and state information. After the simulator receives a new flight plan for the plane, it starts executing the plan. Every flight plan has defined its own initial global simulation time t_0 (which is equal to the time when the simulator had received this plan) and all time information contained in the flight plan is considered relative to it.

The plane simulator executes the flight plans in discrete time steps – within the discrete simulation loop. The interval between such two steps/frames (i.e. updates of the state information of the planes present in the system) is given by the current loaded AGENTFLY system configuration.

The simulator maintains up-to-date information about the states of all airplanes currently present in the system. The state information stored in the data structures of the simulator always contains the real state of the airplanes at the moment of the last update. Let's say that the global simulation time of the last update is t and the state of each airplane $\sigma(t)$ describes its position, direction, normal and velocity. For each plane, the plane simulator also knows its current position in the corresponding flight plan. By the position in the plan, we mean the current segment, the current element and the relative time from the start of the element.

When the next update occurs, the global simulation time will be $t' = t + \Delta t$, where Δt is the time step duration between two successive updates. The state information and the positions of the airplanes in their respective flight plans has to be updated to new values. The state of each airplane is updated by executing the portion of its flight plan corresponding to the time Δt. During the executions of flight plans, the positions of the airplanes in the flight plans are adjusted accordingly. When the update ends, the new position of each airplane will possibly refer to a new segment, element and/or relative position in the element. The segments and elements, which were finished during the update, are removed from the flight plans. If during the execution of the flight plan of some airplane the end of plan is encountered, the airplane is removed from the simulation system.

The time value Δt may be, but doesn't need to be, equal to the real time passed between two updates (in that case, Δt is equal to the update interval); it even does not have to be constant during the simulation. By changing the Δt value, it is possible to change simulation speed; the simulation can be sped up, slowed down or even stopped. Our simulator allows to perform the simulation using various speeds depending on the load of used hosts in the system suitable for fast off-line planning or validation as described in the Section 3.

For obvious reasons, only the parts of the plan, which were not simulated yet (the "future" parts of the plan), can be changed. Let's say we want to change a flight plan (i.e. replan it) and the part be changed (by replacing it with a new plan) is specified by some element marking its start. The replanning will be performed in the way that the portion of the original plan from the marker element to the end of the plan will be replaced by the new plan.

5.2 Airplane Model

In AGENTFLY system, airplanes are modeled as the mass points, which move along a previously planned trajectories. The state σ of an airplane in given time t is defined by the following parameters: the center of mass of the airplane $\overline{x}$, direction vector $\overline{d}$, normal vector $\overline{n}$ and velocity v. The direction vector $\overline{d}$ corresponds to the flight direction. The vector $\overline{n}$, normal vector or *up-vector*, is perpendicular to the direction vector and always aims upwards. Let us denote vector $(\overline{d} \times \overline{n})/||\overline{d} \times \overline{u}||$ (an unit vector perpendicular to vectors $\overline{d}$ and $\overline{u}$) as $\overline{w}$; then, the quadruple $\langle \overline{x}, \overline{d}, \overline{w}, \overline{n} \rangle$ defines the local coordinate system bound to the current state of the plane, as opposed to the world coordinate system $(\overline{O}, \overline{x}, \overline{y}, \overline{z})$, see Figure 4. The plane defined by vectors $\overline{d}$, $\overline{w}$ and passing through $\overline{x}$ is referred to as *plane of flight*.

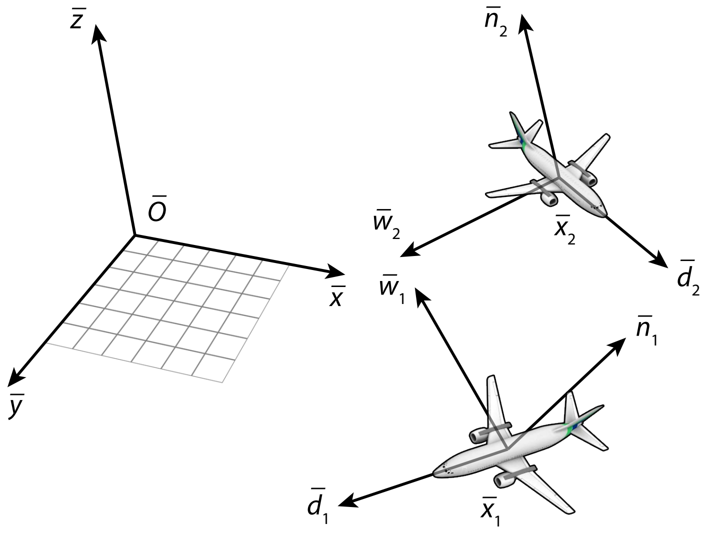


Figure 4: World and local coordinate systems

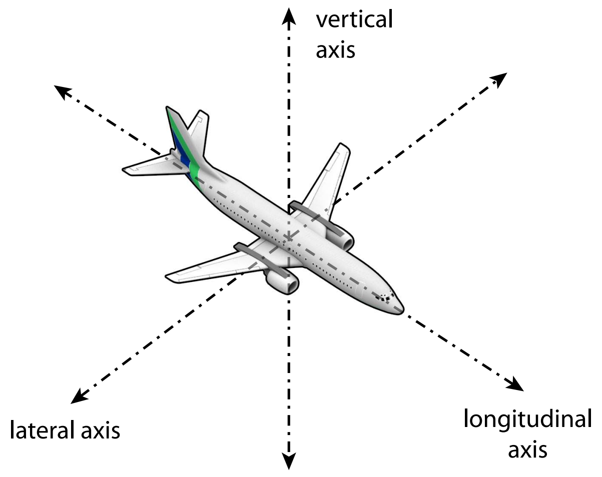


Figure 5: Axes of the airplane

This definition of local coordinate systems of the airplanes allows us to specify the flight plans for the airplanes in a very simple way (the flight plans are described in detail in the Section 4.2). However, it is important to note that this coordinate system is linked with the motion of the airplane and that the directions of vectors $\overline{d}$, $\overline{w}$, $\overline{n}$ do not necessarily correspond to the directions of longitudinal, lateral and vertical axes of the airplane (see Figure 5). As opposed to simplified mass point model, the real airplane has a specific geometry and shape of wings. For example, the lift and drag forces affect the airplane movement thus the direction of flight of the airplane is not identical to its main (longitudinal) axis, but there is nonzero angle α (frequently referred to as the angle of attack). Also, during turns, the airplane is affected also by the centrifugal force, which forces the airplane to turn around its longitudinal axis; the angle between the plane of flight and the plane of the wings of the airplane is called bank angle β. See Figure 6.

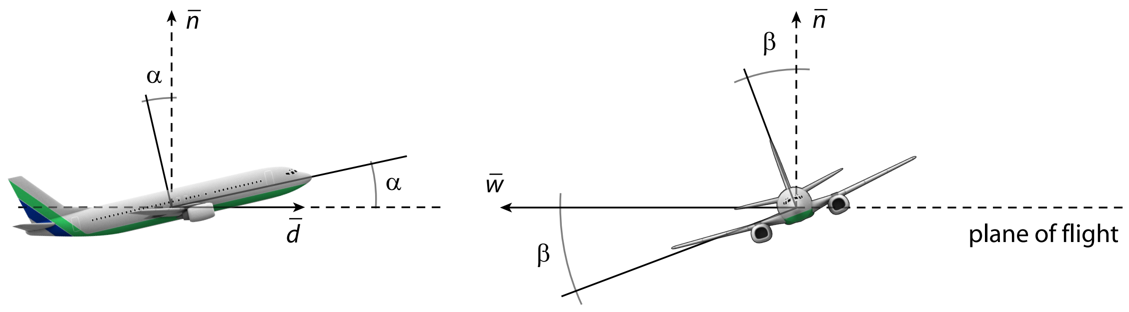


Figure 6: Angle of attack (α) and bank (β) angles

The five main forces, which affect the movement of an airplane during the flight, can be expressed by the following simplified formulas:

- *gravity* $G = mg$,
- *lift* $L = \frac{1}{2} C_L A \rho^2$,

- *drag* $D = \frac{1}{2} C_D A \rho^2$, where $C_D = C_{D_{min}} + \frac{C_L^2}{\pi \cdot ar \cdot 0.75}$,
- *thrust* T, and
- *centrifugal force* $C = \dfrac{mv^2}{R}$, present during turning.

The meaning of the parameters mentioned above is as follows: m is the plane weight, g is the acceleration of gravity, C_L is the coefficient of lift, C_D is coefficient of drag, A is the area of wings, ar is the aspect ratio of wings, ρ is the air density, v is the airplane velocity and R is a radius of turning. Our airplane model is very simple; we do not use the aforementioned formulas as the equations of motion to describe the Newtonian airplane dynamics, but we use them only to determine the magnitudes and directions of the forces affecting the airplane. This information is then utilized for getting the pitch and roll angles of the plane and for computing the thrust force. The pitch (angle between the longitudinal axis of an airplane and xy plane) and roll (angle between the lateral axis of an airplane and xy plane) angles are used for the simulation of the radars (since the area of the airplane profile visible from the ground depends on the banking of the airplane). From the magnitude and direction of thrust we estimate the fuel consumption.

Our model neglects certain more complicated aspects of the airplane geometry, such as flaps or landing gear. Furthermore, in the current version of the simulation system, we assume the airplane weight to be constant during the entire flight – we do not consider the decrease of the weight due to the gradual fuel consumption. We also do not (yet) consider the impact of wind on the velocity and direction of the airplane movement; we assume that the plane performs such maneuvers, that it always follows its planned trajectory and moves at the defined velocity. As for now, we utilize this kind of information only to specify the direction of the airplanes during takeoff and landing, in the way that the airplanes land and take off in the direction against the direction of wind.

6 Flight Path Planning

The planning of the flight path of an airplane is performed by the planner – a component of the pilot agent. The result of the planning is the flight plan, which was described in the Section 4.2. The inputs to the planner are the mission specification for the airplane (represented by the list of way-points, which the airplane has to visit) and the velocity restrictions given by the fixed parts of the flight plan. In addition, the planner must also consider the defined plane airspace. The airspace is restricted by no-flight zones, which the plane must not enter during its flight, and plan the path to avoid them.

Aside from the planning of new flight paths, another important function of the planner is the so-called *replanning*. The replanning changes a part of an existing flight plan while keeping the rest of the plan intact. The replanning is used after the collision avoidance (see the Section 14) and to alter the flight plan to avoid the no-flight zones (described later in this section). The replanning is usually performed by means of inserting of special way-points into a particular segment or segments. The insertion or insertions will cause the splitting of the replanned segment into two or more new segments; these segments are to be planned again, thus creating a replanned flight path. Aside from the ordinary way-points used for the definition of the important navigational points for planning the flight path of an airplane, the *solver* way-points are used during the collision avoidance for applying evasion manoeuvre (see the Section 14.3).

The planning of a flight path proceeds in two phases. In the first phase (path planning), the planner generates an initial flight plan, which passes through all input way-points, Section 6.1. If there are any time constraints associated with the way-points, they are ignored in this phase. The planned path is created to be as short as possible, but still respecting defined airspace thus excluding any defined no-flight zone (Section 4.3). In the next phase – time planning (Section 6.2), certain parameters of the elements generated in previous step are adjusted in such way that the modified flight plan satisfies (if possible) all time constraints.

6.1 The Path Planning

In this phase, the planner generates a detailed flight plan in such way that the flight path will correctly pass through all the way-points as specified in the Section 4.2. The temporal data associated with each way-point are not yet taken into account. For each couple of successive way-points, a segment is generated. The segment is empty, containing no elements. A segment represents the smallest part of the flight plan, which can be planned independently on the other parts of the flight plan. Each segment has several parameters, serving as inputs for planning algorithm (which will fill the initially empty segment with elements). These parameters are the start and end points of the segments with the tangents to the flight plan in these points. The tangents are calculated from the input set of way-points and chosen in such a way that the tangent in the end point of some segment and the tangent in the start point of the next segment will point in the same direction (this property is called geometric continuity or G^1; the planning algorithm assures that the same condition holds also for the elements of the segment; therefore, the planned flight path is always G^1 continuous). During the process of computation of tangents for the way-points it is checked if it is possible to insert there path regarding defined airspace. In other words it is checked how far is the nearest obstacle (ground or no-flight zone) from that point in the selected vector in positive as well as negative direction. If there is not enough space to successfully insert elements respecting airplane type constraints, the tangent in that way-point is adjusted to fulfill that restrictions and have minimal variation from the optimal one.

Initial implementation of fast two-phase A* path finding algorithm (worked only with one no-flight zone kept in octant tree structure) was later replaced by the *manoeuvre-based path-finding algorithm*. The current manoeuvre-based path-finding algorithm is defined by two points (start, destination) and – in contrast with the previous algorithm – also by two vectors (initial direction, target direction).

The manoeuvre-based algorithm incorporates a single A* progressive path planning. As the problem is defined over a continuous airspace and flight plan elements, it is necessary to apply a suitable discrete sampling. The algorithm uses dynamic size of the discrete sampling step depending on the distance from the nearest obstacle from the current position. The size is smaller when it is close to the obstacle. And the size is larger when it is far from the nearest obstacle. Thus it allows to quickly search very large airspace. The number of discrete sampling sizes is given by the planner configuration and is specified for particular world. To be able to find also very small holes in large obstacles the testing distance for appropriate sampling step is given by that size and each new step size must be at most two times smaller than previous one. The algorithm uses path smoothing for removing influences of that dynamic discrete sampling.

In the Figure 7 there is expansion visualization for the 3D world with defined no-flight zones. For that example algorithm searches for the optimal trajectory over more than 30 thousands elements. The discrete sampling of a continuous airspace is done by dividing the plan to several connected search manoeuvres. In other words, the search algorithm searches for the chain of that search manoeuvres connecting the start point and destination point exactly (respecting given initial and target direction vector) with minimal criterion value. Each search manoeuvre is defined by its initial position in airspace and a normalized direction vector. The most of the search manoeuvres also have additional parameters. Each search manoeuvre is also defined by expressions for the calculation of the destination point of the search manoeuvre and a target direction vector. Each search manoeuvre also defines an expression for the calculation of the length of the search manoeuvre which is used for building criterion function for the path search. The

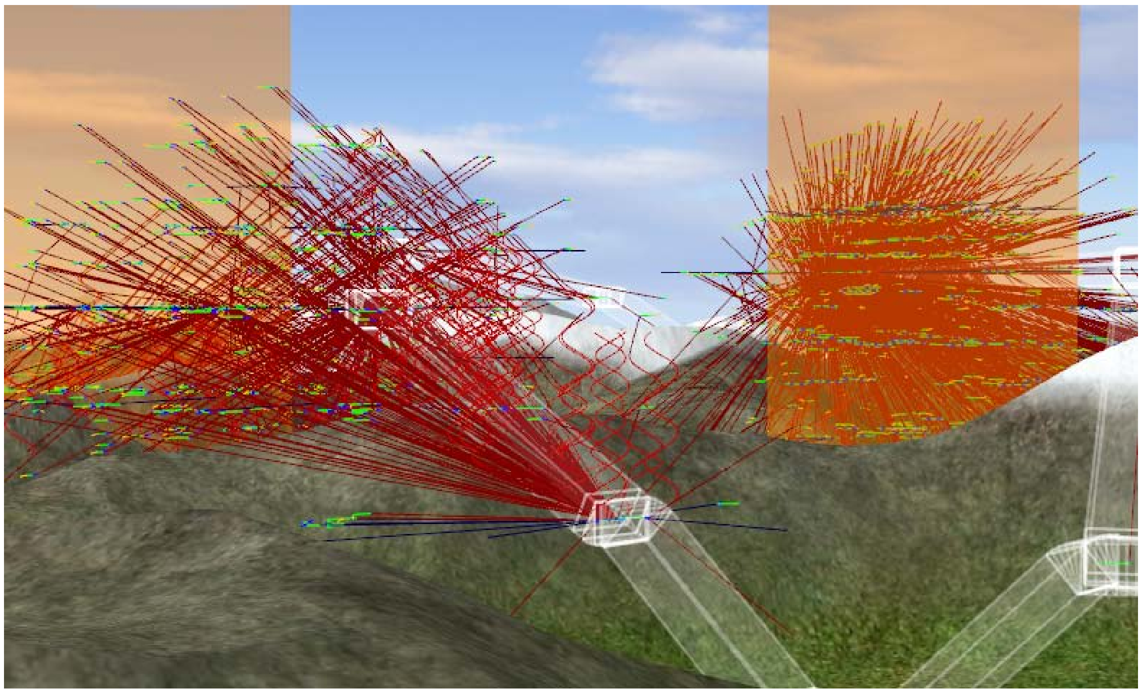


Figure 7: Expansion of the manoeuvre-based path-finding algorithm using dynamic discrete step length in the 3D world with defined no-flight zones.

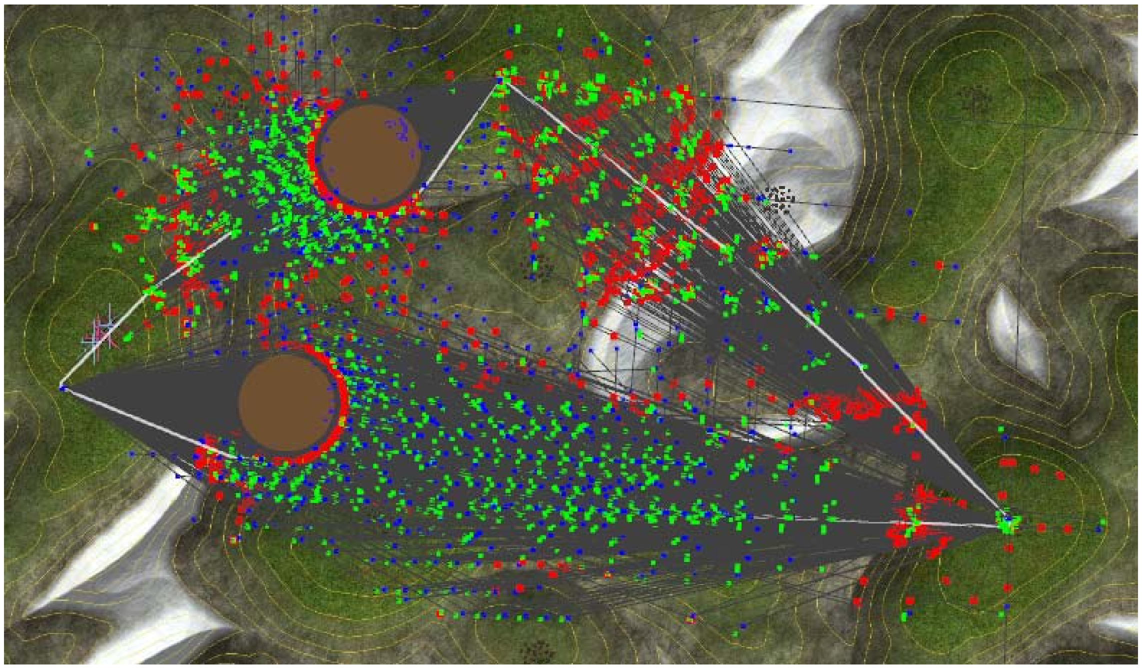


Figure 8: Tested lines for intersection with obstacles. The red points shown identified first intersection on the line.

Simple search manoeuvres (Section 6.1.1) correspond to types of elements of the flight plan and their parameters are described by constants. When using only simple constants, the existence of the solution is not guaranteed because of discrete sampling and the requirement to reach the exact destination point and target direction vector. Therefore it is necessary to extend the set of simple search manoeuvres with combined search manoeuvres (Section 6.1.2). They consist of one or more simple search manoeuvres whose parameters can be determined exactly. For example, for a combined search manoeuvre of flight between two points, all parameters of inner search manoeuvres are calculated so that the destination point of the combined search manoeuvre matches exactly the requested destination point and so that the target direction vectors match.

Expansion of the state space is then carried out by chaining the search manoeuvres according to the A* algorithm, i.e. by the sum of path price g (given by the search criterion function) and estimated price (heuristics) h of the path to the destination point. The expansion discrete step is identified by the end point of previous search manoeuvre from the configuration (as described above). Only such search manoeuvres are added that do not leave given airspace, i.e. they do not run through a ground surface or no-flight zone. This is tested using the line approximations for its defining flight plan elements. In the Figure 8 there are shown all testing lines from the top perspective which are tested for the example in the Figure 7. There were tested several hundreds of thousands lines for the intersections. The red point indicates the first line intersection with boundary of defined airspace. There are intersections around cylindrical no-flight zones and other relates to the intersection with ground surface.

Each newly expanded search manoeuvre is tested if the search algorithm has already visited that position. The test is done by the comparison of the end points and also direction in that end points. The test is done against all closed and not visited but kept in open nodes. To accelerate testing there is used special hashing test supporting also that dynamic discrete sample step. Moreover there is allowed some epsilon-neighborhood to match equality and similar epsilon neighborhood in the direction mishmash. If the match is found, the values $g + h$ are compared and if it is better for the new expansion, the old one is replaced and moved back to open set. The size of the epsilon test depends on the current dynamic discrete factor.

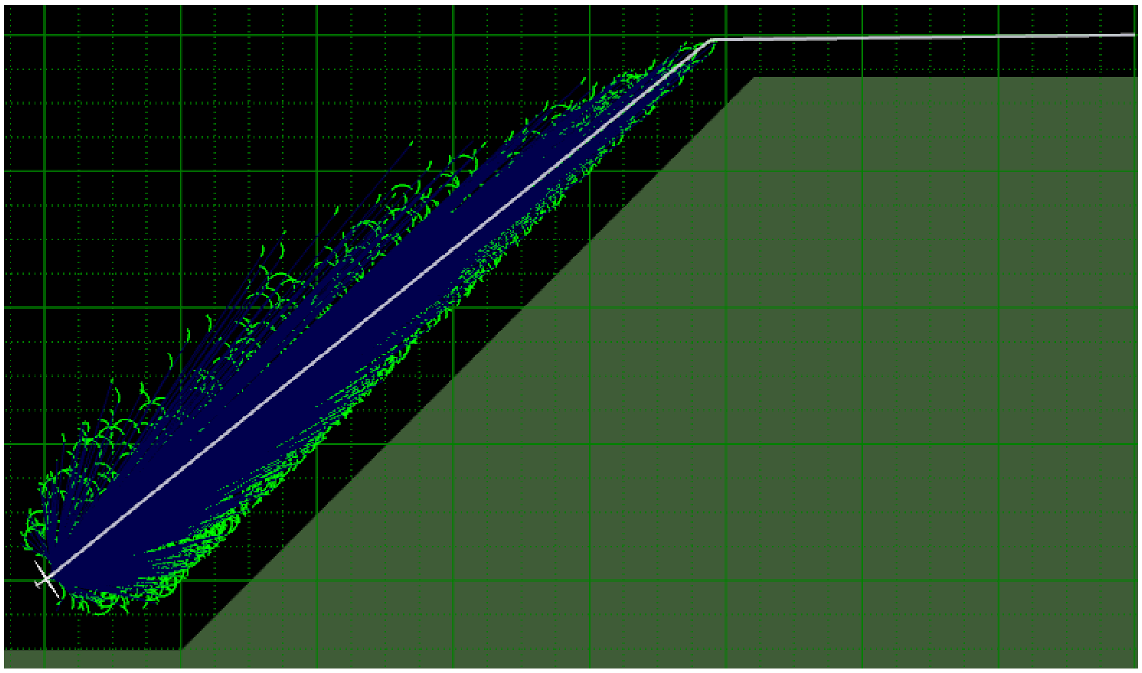


Figure 9: Expansion of the manoeuvre-based path-finding algorithm using dynamic discrete step length in the limited 2D world demonstrating landing scenario.

For each newly connected search manoeuvre, the possibility of replacing this search manoeuvre with a smoothing search manoeuvre is tested. The testing process tries to sequentially replace the path between the previous search manoeuvre and the search manoeuvre being connected. Previous search manoeuvres are tested starting from the first search manoeuvre of the entire planning task to the search manoeuvre directly preceding the one being connected. The particular smoothing search manoeuvre is used only in the case it doesn't intersect any no-flight zone and its length is not greater than the length of the existing path plus the length of the added search manoeuvre that is supposed to be replaced (smoothed). The search smooth manoeuvre is always as short as possible between its end points. Thus this replacement can only provide shorted connection and/or connection with less necessary flight plan elements which leads to better g criterion value.

The prize of path g is calculated as the length of manoeuvres including the current one. The value of heuristics h is calculated as the length of the theoretical combined shortest smooth search manoeuvre between two points, from the end point of the current manoeuvre to the global destination point. The use of smooth search manoeuvre as a heuristics predictor provides the best satisfiable heuristics function for the search problem. It also covers the situation when current end point is too close to the destination but has wrong direction which doesn't allow to reach the final point in that small distance due to the constraints given by the plane model.

The algorithm terminates upon removing the search manoeuvre reaching the final destination (both in the position and direction without any tolerances) from the OPEN list or upon emptying the OPEN list. If the path is found, i.e. if the search manoeuvre most recently removed from the OPEN list reached the destination point, the actual path is generated using a backward trace, where simple flight manoeuvres are converted to flight plan elements. The algorithm takes always the search manoeuvre with the lowest value $g + h$ from the OPEN list.

The implementation of the search algorithm and testing line methods for airspace definition is very efficient and uses all known algorithm programming techniques. The whole searching process for the shown examples takes only couple of milliseconds. Next expansion example in the Figure 9.

6.1.1 Simple Search Manoeuvres

This section briefly presents all simple search manoeuvres used within manoeuvre-based path-finding algorithm:

- *straight flight* – basic search manoeuvre following a straight line,
- *horizontal turn* – search manoeuvre following a section of a circle lying in a horizontal plane,
- *climbing/descending* – search manoeuvre following section of a circle lying in a vertical plane,
- *spiral* – climbing/descending search manoeuvre following a spiral; the x and y coordinates of the start and end points of the search manoeuvre are identical, the only coordinate that differs is z (the altitude).

6.1.2 Combined Search Manoeuvres

There are also several combined search manoeuvres used:

- *combined flight search manoeuvre towards the desired flight level*,
- *combined flight search manoeuvre connecting two points* – consists of three to seven search manoeuvres (see Fig. 10, 11 and 12):
  - turn towards the destination point (horizontal turn search manoeuvre),
  - climb/descent towards the destination point (climbing/descending search manoeuvre),
  - straight flight towards the target flight level (straight search manoeuvre),
  - spiral flight to the desired flight level (spiral search manoeuvre),
  - straight manoeuvre to the target flight level (straight search manoeuvre),
  - altitude correction to the target flight level (climbing/descending search manoeuvre),
  - optional correction towards the destination point and target direction using spiral search manoeuvre (combined search manoeuvre connecting two points),
  - turn towards the destination point (horizontal turn search manoeuvre),
- *combined smoothing search manoeuvre* – serves as an element connecting two points with arbitrary direction vectors; consists of one to three manoeuvres:
  - climb/descent towards the desired flight level (combined search manoeuvre towards the desired flight level),
  - flight between two points on different flight levels (combined search manoeuvre connecting two points),
  - climb/descent towards the target direction (inverse combined search manoeuvre towards the desired flight level).

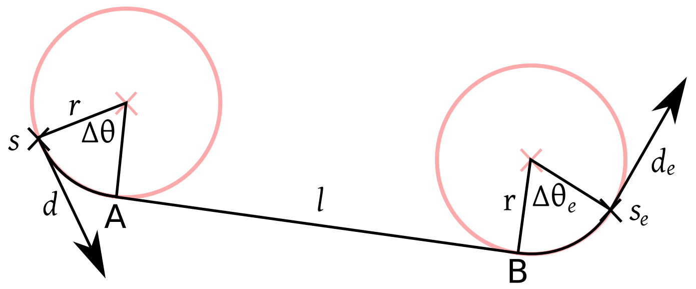


Figure 10: Combined search manoeuvre between two points on the same flight level

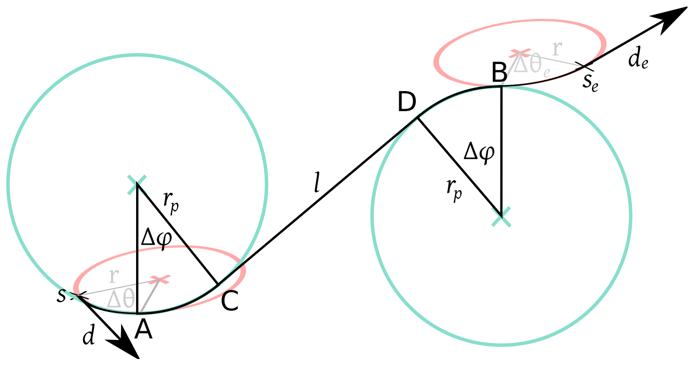


Figure 11: Combined search manoeuvre between two points on different flight levels

6.2 The Time Planning

The flight plan, which is the output of the first phase of planning (spatial planning), constitutes only the initial sketch of the flight route of an airplane; it does not take into account the time relations along it and it ignores all the temporal data assigned to the way-points. In the next phase of planning, the previously planned segments are adjusted so that the resulting flight plan conforms to the time constraints. As a result, the time of flight through each segment will correspond to the time constraints defined in the start and end way-point belonging to the segment.

For purposes of the time planning a transformation that converts rather complex spatial plan to a simpler sequence of so called time constraints was used, i.e. from $\mathbb{R}^4$ to $\mathbb{R}^2$, where three dimensions become one that corresponds to lengths of elements (manoeuvres).

The fore-mentioned time constraints can be of two types: point constraints and section constraints. More specifically, there are the following six types:

- speed point constraint $\{v_c, s\}$,
- minimal time point constraint $\{t_{min}, s\}$,
- maximal time point constraint $\{t_{max}, s\}$,
- free point constraint $\{s\}$,
- constant speed section constraint $\{l, s\}$,
- free section constraint $\{l, s\}$.

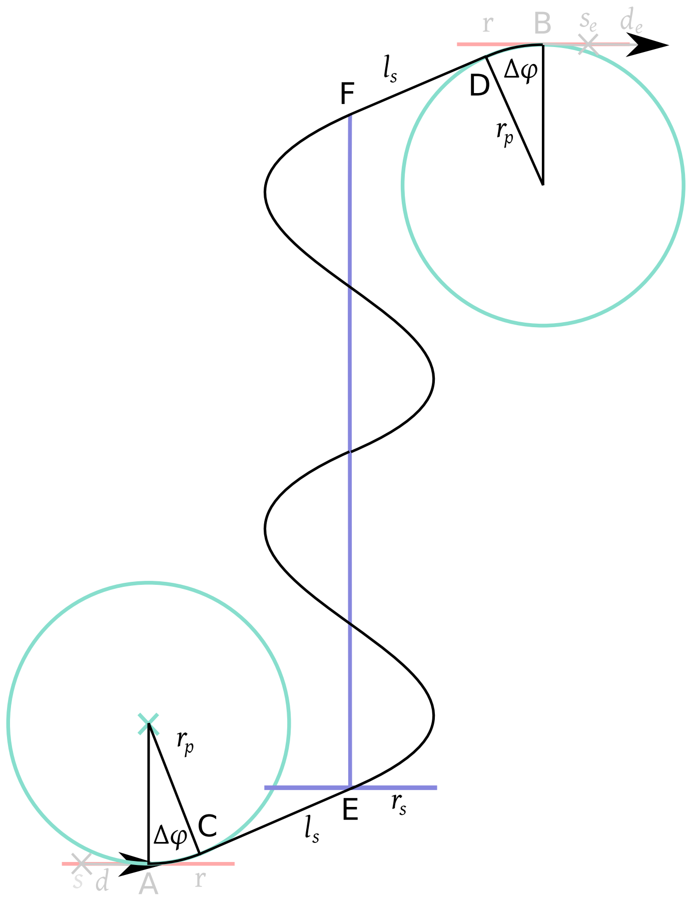


Figure 12: Combined search manoeuvre between two points on different flight levels using a spiral

Point constraint of speed defines that the given point s in the plan must be flown through at the speed of v_c. Minimal time constraint defines that the point s must be flown through at time t_{min} at the earliest. Similarly, maximal time constraint defines that the point s must be flown through at time t_{max} at the latest. Free point constraint has no theoretical purpose and is mentioned here for the sake of completeness.

Section constraints are applied homogenously to a section of the plan starting with a point s and of length l. Constant speed section constraint prohibits to change the speed within the particular section of the plan. It's obvious that this constraint is used for flight plan elements other than straight ones which do not allow speed changes (turns and spirals). Free section constraint is used for filling space with no constraints (for straight elements). An example sequence of time constraints and the source plan can be seen in Figure 13.

A sequence of time constraints generated this way is subsequently used as the input for the actual time planning algorithm. The task is therefore explicitly defined by this sequence.

Similarly to the algorithm of spatial planning (Section 6.1), the time planning algorithm is based on particular elements' chaining. In the case of spatial planning these elements were search manoeuvres, in case of time planning they are time plan elements, or in short time elements. In contrast to spatial planning, time elements are not expanded using algorithms for state space exploration (such as A*). Instead, they are only suitably chained one by one, with pre-calculated

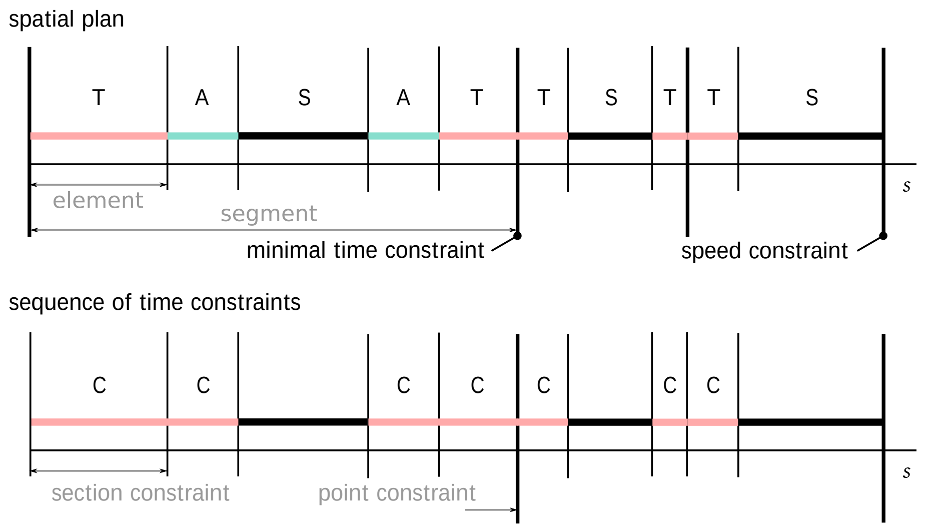


Figure 13: Example of the sequence of time constraints and its respective source spatial plan. (T - turn, A - climbing / descending, S - straight flight, C - constant speed)

parameters. In contrast to the spatial planning, it can therefore never happen that a single manoeuvre is expanded to many potential successors. Elements of the time plan always have just a single successor (expect for the last one which has no successor at all).

Time elements, as much as spatial search manoeuvres, have a start point $\overline{s}$ (in the forementioned space $\mathbb{R}^2$, where point $\begin{pmatrix} t \\ s \end{pmatrix}$ has one time dimension t and one space dimension s) and initial speed v. Similarly to spatial search manoeuvres, most time elements have additional parameters and there are also defined expressions for calculation of the destination point $\overline{s_e}$ and target speed v_e.

The time planning algorithm operates in so called sequences of time constraints. Each sequence contains an arbitrary number of time section constraints and one point time constraint at the end. Point constraints inside the sequence are not permitted as they would break the sequence into two. The point constraint at the end defines the type of the sequence and thus also the planning method. An example sequence (see Figure 13) depicts two sequences. The first consists of five section constraints (2 constant speeds, 1 free and 2 constant speeds again) and a minimal time point constraint at the end. The sequence is therefore a minimal time sequence. The other one also consists of five section constraints (1 constant speed, 1 free, 2 constant speeds a 1 free) and is a speed sequence.

Sequences are planned in successive steps. The algorithm connects the time elements so that the section constraints are fulfilled and so that the point constraint at the end of the sequence is fulfilled as closely as possible. Time elements are connected continuously speed-wise and smoothly direction-wise.

If time elements cover the entire sequence of time constraints, or if planning of all sequences is over, the time plan has been generated and can be transformed to a spatial plan, i.e. planned increments and decrements of speed are added to the spatial plan. Optionally, slow-down spirals are inserted. At that point, the flight plan is completed and ready for use.

7 AGENTFLY Architecture Overview

The AGENTFLY system is fully written in JAVA language. For the visualization purposes it requires external libraries which are open source and are available for several host platforms. Using JAVA language the system can be easily spread on more host computers with different operating systems. The system consists of several components, see Figure 14:

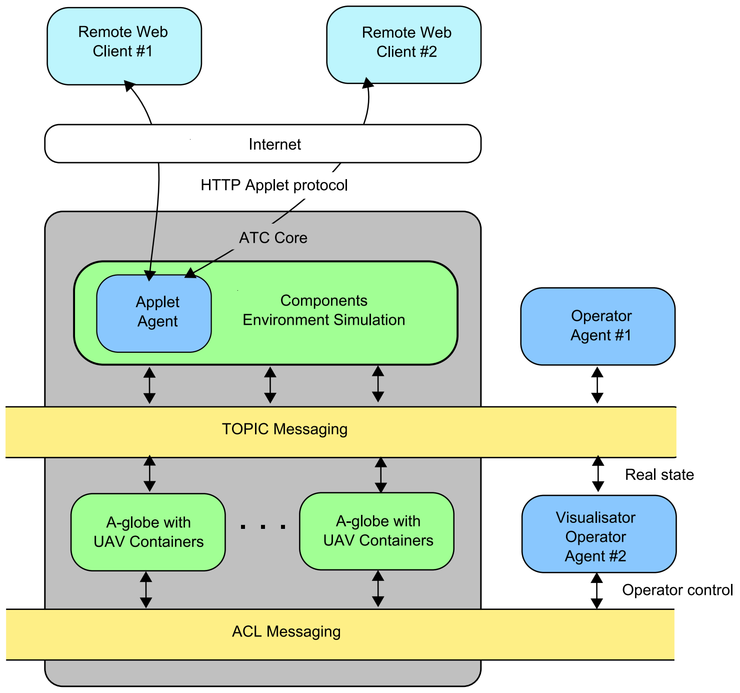


Figure 14: AGENTFLY system structure overview.

- **AGENTFLY Core System** – mandatory component of the system that is responsible for aircraft simulation and airways planning. All parts of this component are represented by agents which are running on **$\mathcal{A}$-globe** JAVA multi-agent platform [1, 13]. This component also provides data for connection several number of Operator agents which provide real-time visualization of the system state as well as human system interfaces. The architecture of this component is described in the Section 9.

- **Remote Web Client** – optional component allowing remote user to connect and interact with the AGENTFLY core system. In current version the user can only display information which he needs. It means that there is not allowed any feedback from the user back to the AGENTFLY system, e.g. user cannot modify way-point plan (mission) of any aircraft or insert new flight to the system. Such functionality is provided by the Operator Agent. The Web Client is simply started by the entering URL address of the AGENTFLY core system to the internet browser. The client is written in JAVA built up on the JOGL libraries (Java binding for OpenGL [5]) which are used for accessing graphics 3D acceleration. The AGENTFLY core system uses Java Web Start application for the client loading and starting. Before a user can use all features of the remote WEB client, he needs to be successfully logged with valid username and password. For minimizing network traffic between the remote client and the core system combination of HTTP and special binary protocol is used. Remote client authentication procedure is secured using one-time hashes for password validation. If it is necessary the whole data communication can be secured using asymmetric cryptography but it comes with higher processor load requirements. The overview of user interface and data provided via remote client is in the Section 10.

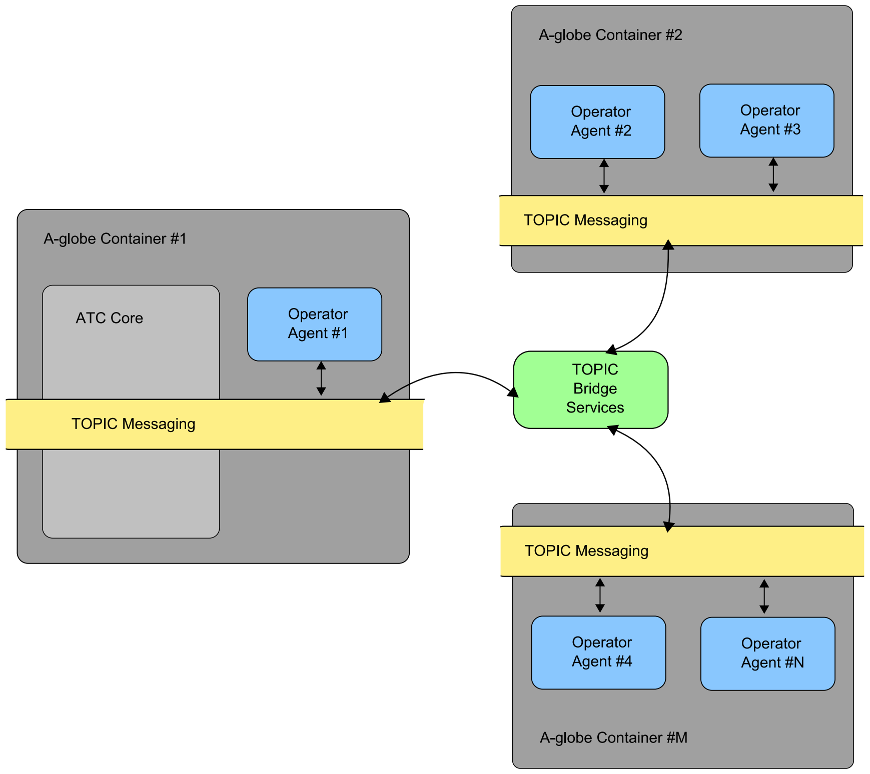


Figure 15: Example of the Operator Agents dissemination setup.

- **Operator Agent** – optional component that can provide real-time system state with all important information in a 3D/2D environment to the user. Provided information layers depend on its current configuration. There can be running more such agents simultaneously providing different information with proper access level rights. The detailed description of the visualization components of the operator agent is provided in the Section 11.

  Besides the visualization the Operator Agent is also able to send user commands back to the system or can be directly connected to the specific aircraft. Thus it is able to provide human-system interface to the user. The user can then manage aircraft's way-point plan (mission) or manage defined forbidden no-flight zones. It is regular **$\mathcal{A}$-globe** agent which receives requested real state information using topic messaging. Direct connection to the aircrafts is realized via operator control channel on the top of the standard agent communication language (ACL) messages. Detailed description of the human-system interface of the Operator Agent is in the Section 12.

All system components can run on the same host computer or the system can be spread on more hosts for simultaneous simulation of huge number of aircraft. Components requirements for host machine are specified in the Section 20.

The Figure 15 presents AGENTFLY system configuration with more Operator Agents running. Each $\mathcal{A}$**-globe** container can run in own JVM on different host computer. It is possible to connect Operator Agent providing visualization and/or operator function remotely. Connection of remote container to the topic messaging is allowed thanks to its bridging by the set of $\mathcal{A}$**-globe** topic bridge services. Only requested topic layers are transferred to the remote containers.

8 Integration of External Data Sources

The AGENTFLY system has been integrated with real world data. External data are taken from free internet resources of various GIS data for the area of the United States. Data from several web sites were parsed, filtered and merged together to serve as a database for our simulation system.

- **Landsat7 Images** – a mosaic of Landsat7 images at the maximum resolution of approximately 50 meters per pixel was used as an underlying ortophoto map of the United States. The source data for this layer were collected from `http://onearth.jpl.nasa.gov`. For purposes of performance optimization and data flow reduction, the original satellite images had to be split into uniform tiles of 512×512 pixels. The same image mosaic was generated in multiple resolutions to allow seamless zooming in and out of the virtual map. Since the data for this layer consist of about 1.7 gigabytes of JPEG-compressed images covering the entire area of the United States, a sophisticated system of switching between image tiles and multiple resolution representations was the only and the best way to manage such amount of data. See Figure 16.
- **State Boundaries** – detailed vector shapes of 50 U. S. state boundaries obtained from `http://seamless.usgs.gov`. Extensive post-processing of the data had to be carried out as the source data contained a lot of shape redundancies and duplicities. See Figure 16.

Airports – a set of more than 650 U. S. airports, including their names, GPS coordinates and the corresponding average numbers of enplanements per year obtained from the site `http://seamless.usgs.gov`. The size of the airport icon reflects the average number of enplanements per year. See Figure 17.

No-flight Zones – for the purposes of the simulation we decided to use U. S. powerplants to act as the no-flight zones. A set of more than 80 U. S. powerplants, including their names and GPS coordinates was derived from the data available at `http://geonames.usgs.gov`. See Figure 17.

Cities – a set of more than 24 thousands U. S. populated places, including their names, GPS coordinates and the corresponding population obtained from `http://geonames.usgs.gov`. The size of the city icon reflects the size of the population. See Figure 19.

Highways – a set of more than 26 thousand major U. S. highway segments derived from the data available at `http://seamless.usgs.gov`. See Figure 19.

Real Air Traffic – nearly real-time data of the current air traffic over selected U.S. airports updated periodically from internet sources. This data layer is used as input for testing the NFZ-based non-cooperative collision avoidance algorithm as described in 18.

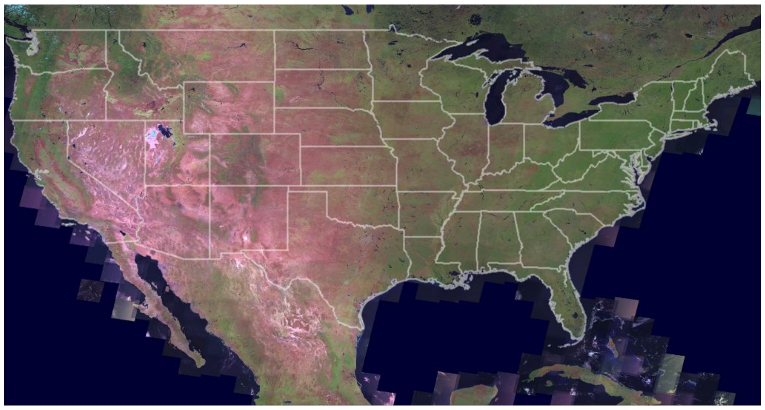


Figure 16: GIS data layers: Landsat7 images and U. S. state boundaries

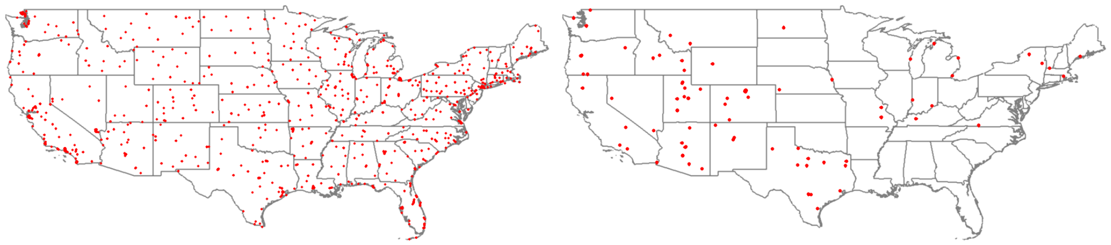


Figure 17: GIS data layers: airports (left) and no-flight zones (right)

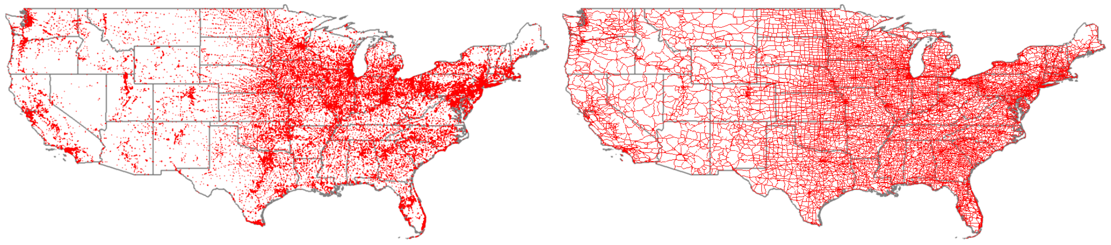
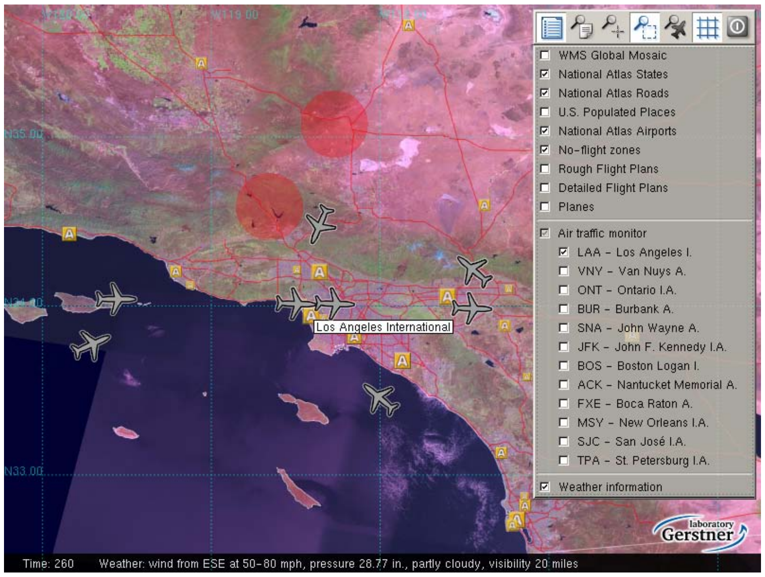


Figure 18: GIS data layers: cities (left) and highways (right)



- [ ] WMS Global Mosaic
- [✓] National Atlas States
- [✓] National Atlas Roads
- [ ] U.S. Populated Places
- [✓] National Atlas Airports
- [✓] No–flight zones
- [ ] Rough Flight Plans
- [ ] Detailed Flight Plans
- [ ] Planes
- [✓] Air traffic monitor
  - [✓] LAA – Los Angeles I.
  - [ ] VNY – Van Nuys A.
  - [ ] ONT – Ontario I.A.
  - [ ] BUR – Burbank A.
  - [ ] SNA – John Wayne A.
  - [ ] JFK – John F. Kennedy I.A.
  - [ ] BOS – Boston Logan I.
  - [ ] ACK – Nantucket Memorial A.
  - [ ] FXE – Boca Raton A.
  - [ ] MSY – New Orleans I.A.
  - [ ] SJC – San José I.A.
  - [ ] TPA – St. Petersburg I.A.
- [✓] Weather information

Time: 260 Weather: wind from ESE at 50–80 mph, pressure 28.77 in., partly cloudy, visibility 20 miles

Figure 19: Real air traffic over L.A. International Airport

9 Core System Description

Agent-based AGENTFLY core system encapsulates one *server component* running components for environment simulation (described in the Section 9.1) and one or more *𝒜-**globe** platforms* for running UAA containers, Figure 14. One of the *𝒜-**globe** containers* in each JVM is used as a registration unit for starting *UAA container* (described in the Section 9.2). When AGENTFLY system is used for the simulation of huge number of aircrafts it is highly recommended to use more host computers running 𝒜-**globe**. There is no reason to start more JVMs on the same host. It leads to the higher resource requirements consumed by JVM processes. If there is no 𝒜-**globe** platform registered, the UAA containers are automatically started on the same host with the server.

In the situation when AGENTFLY system is started on the two or more host computers, it is useful to run *server component* separately on the one computer and rest computers has one *platform 𝒜-**globe** container* on each of them. The number of running UAA containers on the more registered platforms is proportionally split among them. By this way the AGENTFLY system balances the overall load among all registered hosts.

9.1 Server Components for Environment Simulation

The *server component* of the AGENTFLY core system is sole central element of the system used for the environment simulation. It simulates positions of aircrafts and other objects in the simulated world, aircraft hardware, weather conditions, communication ranges given by range of board data transmitters, etc. When the proposed distributed agent system for flying on collision-free airways is used to control real aircraft assets, this *environment simulation components* will be removed and will be replaced by real sensors and actuators of the assets.

One of the simulation components is also responsible for acquiring and fusion of information about all planes with freely available geographical and tactical data sources (see section 8) and provides them to both *remote WEB client* (section 10) and *operator agents* (section 11). It works also as a *scenario player* which takes care of creation of new aircraft with a rough plan mission in specific time moment.

The *server component*, figure 20, consists of several agents:

- **Configuration Agent** – handles configuration of the AGENTFLY system. User can easily select current configuration from the combo box in its user interface. It loads initial selected configuration and distributes all necessary information to the other agents. The *configuration agent* allows user to change the configuration within running AGENTFLY system. If the AGENTFLY system is running on more host computers, the configurations are distributed from the computer where the configuration agent is. User of the system can easily modify configuration and restart the simulation again without copying changed files to all other computers.
- **Entity Manager Agent** – administrates connected 𝒜-**globe** platforms and running aircraft containers, starts new or removes existing planes, gives initial flight mission to the plane. Depending on the current system configuration *scenario player* modules are loaded. Each module can start new or remove existing aircraft, and provide also its initial configuration. There are several prepared scenario players:
  - *general* – plays scenario exactly as specified in the scenario script file. It can have more instances simultaneously playing different script files,

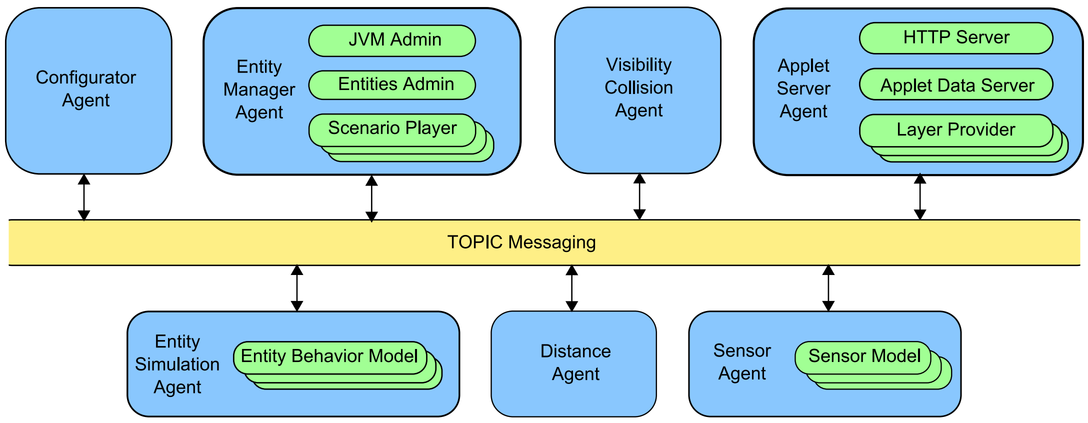


Figure 20: Components for environment simulation

– *user startup* – has graphical user interface allowing user to generate new planes in the simulated world manually,

– *experiment* – special scenario player used for performing huge experiments. The player automatically starts more repetitive runs and acquires requested measure values to the experiment output.

The *Entities Admin* works as a load balancer among connected host computers where the AGENTFLY core system is running. The new aircraft container is started on the registered **$\mathcal{A}$-globe** platform where there are minimum running plane containers. Depending on the configuration the selected scenario can be started in the off-line planning, normal or validation simulation phase (described in the Section 3).

- **Entity Simulator Agent** – computes true position of the aircraft in the simulated world. Each entity in the simulated world has assigned one *entity behavior model* module. It is responsible for the entity physical location and behavior. For the aircraft model (Section 5.2) we have created `AircraftBehavior` which contains physical model for jet aircraft and holds its current flight plan and state. When a plane *pilot agent* changes some part of the flight plan, the change is propagated via *plane agent* to its *behavior model* running in *plane simulator agent* by the difference flight plan update. On the other side the module can be asked for the plane position in current simulation time. The module is also responsible of handling collision situations, e.g. free fall simulation. The details about physical modeling of the aircraft can be found in the Section 5.2.

- **Distance Agent** – counts euclidian distances between every pair of existing aircrafts using the positions generated by the *plane simulator agent*. It also provides euclidian distance in XY plane and other statistical values, such as minimal separation from each plane and overall minimal separation among all aircrafts.

- **Visibility Collision Agent** – prepares **$\mathcal{A}$-globe** visibility updates [13] for controlling communication restrictions among all aircrafts. It also detects whether there is physical collision between flying aircrafts. If the collision is detected the aircraft behavior model is notified about it. The airplanes that had collision with other objects are uncontrollable and they go down to the ground. Falling aircraft can endanger any plane which flies under it.

- **Sensor Agent** – represents all radar sensors on aircraft boards. *Plane agents* can register a specific sensor within it. The *sensor agent* sends radar information to the registered agents depending on the sensor characteristics and aircraft positions and orientations.

- **Applet Server Agent** – runs HTTP server, Applet Data server and includes all external data layer providers. It provides communication interface between AGENTFLY agent system and the *remote WEB client* (described in the Section 10). For allowing access to the AGENTFLY system from the networks shielded from the internet by the proxy servers the HTTP server listens on standard TCP/IP port 80 and Applet Data server listens on the port 443 normally used for HTTPS protocol (this port is typically tunneled in such networks). AGENTFLY system generates cache and proxy control headers for controlling cache content validity. The *Applet server agent* has running data layer providers which provide external and internal content to the *remote clients*. All integrated external data sources are described in the Section 8.

All server modules (agents) communicate together using $\mathcal{A}$-**globe** ***topic messaging*** described in [13]. All used topics are defined in the project class `aglobex.simulation.global.Server-ServerTopicConstants` where the description of the topic content can be also found. The communication between *server agents* and platform hosted agents also utilizes ***topic messaging*** due to its easy usage without any complicated addressing. In the project class `aglobex.simulation.global.-ClientServerTopicConstants` there are defined all topics used for client-server communication.

9.2 $\mathcal{A}$-globe with UAA Containers

As described in the Section 9, first connected $\mathcal{A}$-**globe** container in each JVM is used as the registration unit for that JVM. There can be no agent running in this container. The container acts as a control bridge between *entity manager agent* and the local JVM where it is running. Using its system services new *UAA container* can be started or removed. In one $\mathcal{A}$-***globe*** *platform* (inside one JVM) there can run many *plane containers* or there can be no *plane container* at all. If there are more JVMs connected to the system, new UAA containers are automatically balanced among them. Runtime configuration (number of computers and JVMs) doesn't depend on configuration files. The system can be easily started in any configuration by modifying only starting batches. Design of the *UAA container* is shown in the Figure 21.

The UAA containers interact with the server components responsible for the environment simulation via $\mathcal{A}$-**globe** topic messaging. There is also interaction among UAA containers used for cooperative deconfliction negotiation using ACL messages among agents. Operator control channel between *Operator Bridge* and *Operator Agent* is direct agent communication too.

9.2.1 Plane Agent

The *Plane Agent* provides interface to higher-level plane functions. It provides following functionality:

- *Configuration* – provides configuration including UAA type descriptor and initial mission specification with its way-point plan – time constrained or no constrained (see Section 6. The type descriptor for the plane holds information about plane capabilities, such as minimal and maximal flight velocity, acceleration, declaration, minimal turn radius, max angle for changing altitude (flight level), weight, etc.
- *Radar* – sends information gathered from the UAA on-board radar. There are transmitted information about actual position of observed objects. The information is repeatedly updated after each radar scan.

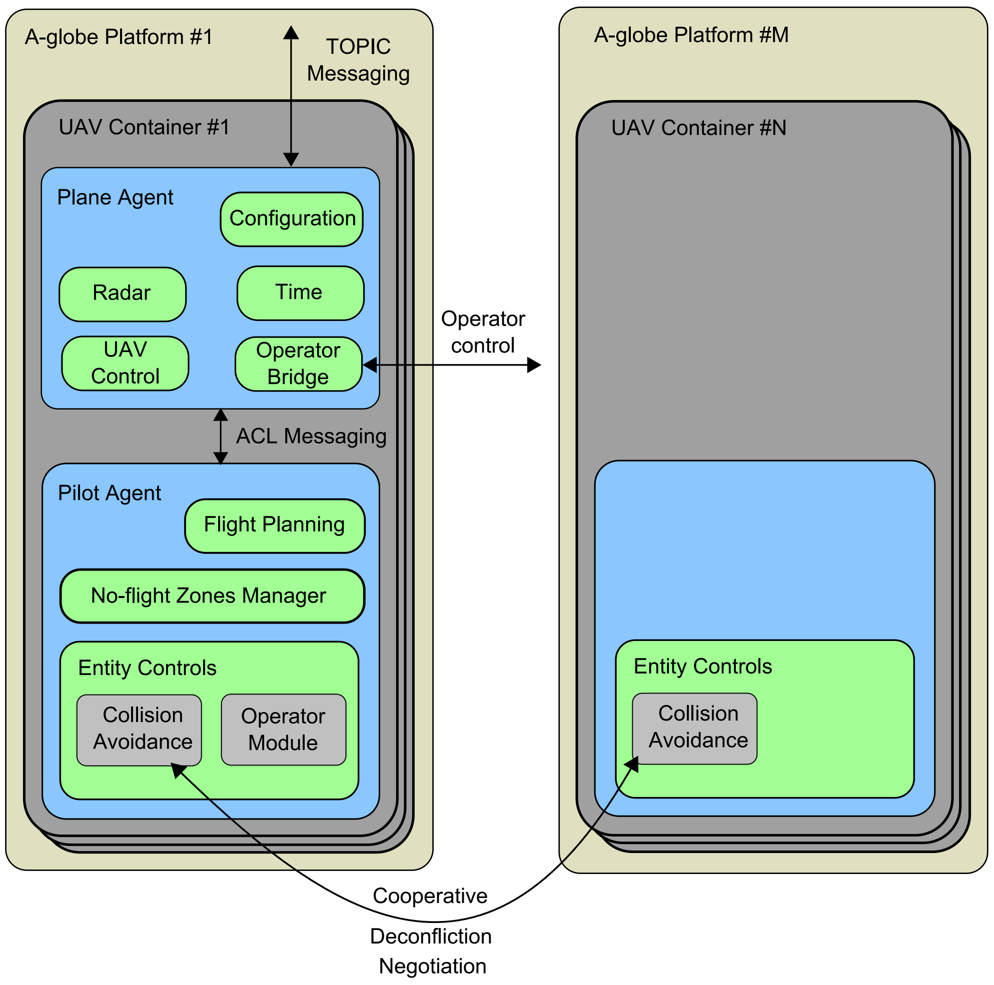


Figure 21: $\mathcal{A}$-**globe** with UAA containers

- *Time* – broadcasts timestamps.
- *UAA Control* – provides flight plan execution and can be requested for the current UAA position and state. The executed flight plan can be changed at any time by sending differential update for the current flight plan. Only future part of the flight plan can be changed.
- *Operator Bridge* – sends debugging information to the specific graphics layers in the operator agent and provides bi-directional communication interface between *operator module* running in pilot agent and appropriate operator user interface.

In the AGENTFLY system the plane agent provides interface to the simulated environment. The plane agent can be replaced by other version which will provide interface to the real UAA sensors and actuators for controlling the UAA motion.

9.2.2 Pilot Agent

The *Pilot Agent* is the main control unit for the UAA. It provides control for its operations. Several subsystems are included:

- *Flight Planning* – is used for the detailed flight plan planning and replanning on the requests generated by other modules. It interacts with *no-flight zones manager*. Its planning must respect all currently known forbidden no-flight zones. Its functionality is described in the section 6.
- *No-flight Zones Manager* – manages the airplane airspace definition, see the Section 4.3. The zones are grouped into three categories depending on their purpose: *world*, *static* and *dynamic*. The octant tree and height-map representation of the zones can be pretty large data structures, thus the zone manager implements sharing of the basic zones representation among all managers running within the same JVM. Only the zones which shape is never updated during one simulation are shared.
- *Entity Controls* – is modular architecture for multi-module entity control. Which modules will be created after UAA start is given by the UAA configuration, but the number of the running control modules can vary during UAA operation. All control modules can generate current flight plan changes, which will be applied depends on the modules priorities. Modules can use radar notifications and communication interface provided by entity control. Currently there are three modules:
  - *Collision Avoidance* – provides collision avoidance ability for the UAA. It processes notifications about new visible object(s) on the radar. Within cooperative mode it tries to communicate with respective *pilot agent* if there is any. It is also responsible for collision detection between its flight plan and the other flight plan part (see section 14).

    Within non-cooperative mode it monitors and predicts future possible positions of all visible objects. Using no-flight avoiding mechanism it tries to replan its flight plan to collision free one, describe in the section 15.

    It is implemented as a *multi-layer collision avoidance* module which allows to use several types of collision avoidance methods simultaneously. It is described in the section 13.
  - *Operator Module* – is a module that allows human operator to change the UAAs mission (way-point plan specification) and edit its static forbidden no-flight zones. The detailed interaction with the human operator interface is described in the section 11.
  - *Group Coordination* – is used for the collective flight, see the Section 16. It does *inner group synchronization* – decides if and where to take a holding orbit to wait for delayed airplanes, performs precise composition and decomposition of the flight formation package and guarantees collision-free paths in this phase. Besides inner group synchronization it provides also *outer group-to-group synchronization* – identifies which rendezvous points should be used in the case of concurrency.

10 Web-based Access

The AGENTFLY simulation state can be accessed using a Java client web application that is via the network connected to the simulation system acting as a server and providing to all its clients regular data updates. This way, a number of users can concurrently observe and interact with the same simulation. Accessing the simulation system via the network is as simple and straightforward as opening a web browser and entering the IP address of the computer on which the $\mathcal{A}$**-globe** simulation system is currently running. More specifically where the Applet Server Agent is running.

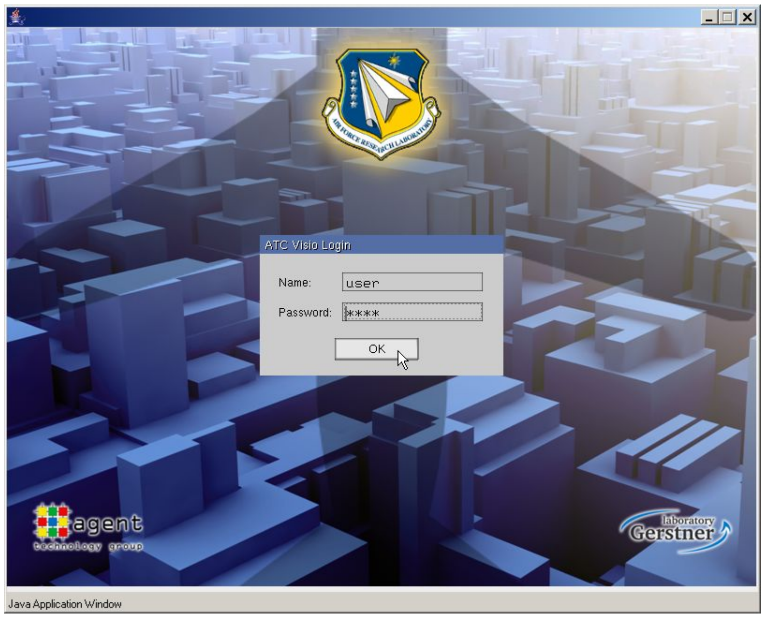
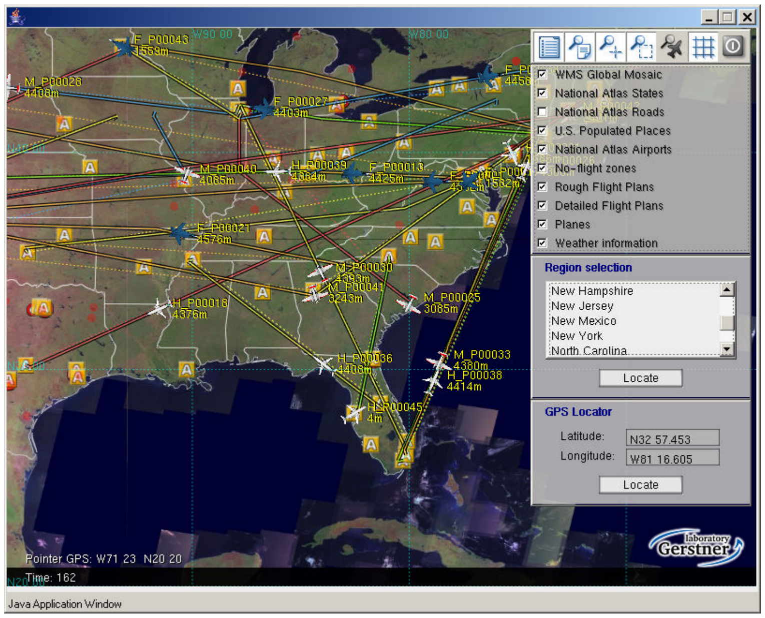


Figure 22: The dialog for authentication of the user to the web AGENTFLY component

Before the user can access the simulation, he needs to log in the system by entering a valid user name and password as shown in the Figure 22.

In the top right hand corner of the screen (see the Figure 23) there is located an icon palette which gives the user access to application controls:

- **Layer Menu** – allows the user to switch on and off various data layers in the main view. There are currently data layers representing all integrated external data as described in the section 8 and the layers presenting the internal AGENTFLY system state presented in the Section 10.1.
- **Zoom to Region** – allows the user o zoom to a specific region (state) by selecting it from the list.
- **Locate GPS** – allows the user to move to a specific location by entering its GPS coordinates.
- **Zoom to Selection** – lets the user zoom to a specific part of the scene by selecting a rectangular area with the mouse.


☑ WMS Global Mosaic
☑ National Atlas States
☐ National Atlas Roads
☑ U.S. Populated Places
☑ National Atlas Airports
☑ No–flight zones
☑ Rough Flight Plans
☑ Detailed Flight Plans
☑ Planes
☑ Weather information

Region selection

New Hampshire
New Jersey
New Mexico
New York
North Carolina

Locate

GPS Locator

Latitude: N32 57.453
Longitude: W81 16.605

Locate

Pointer GPS: W71 23 N20 20
Time: 162

Java Application Window

Figure 23: The AGENTFLY web interface

Zoom to Airplane – allows the user to zoom to a specific airplane by clicking on it.

Toggle Coordinate Grid – allows the user to toggle the GPS coordinate grid.

Logout – allows the user to log out from the system.

The user can also use the keyboard to navigate in the main view:

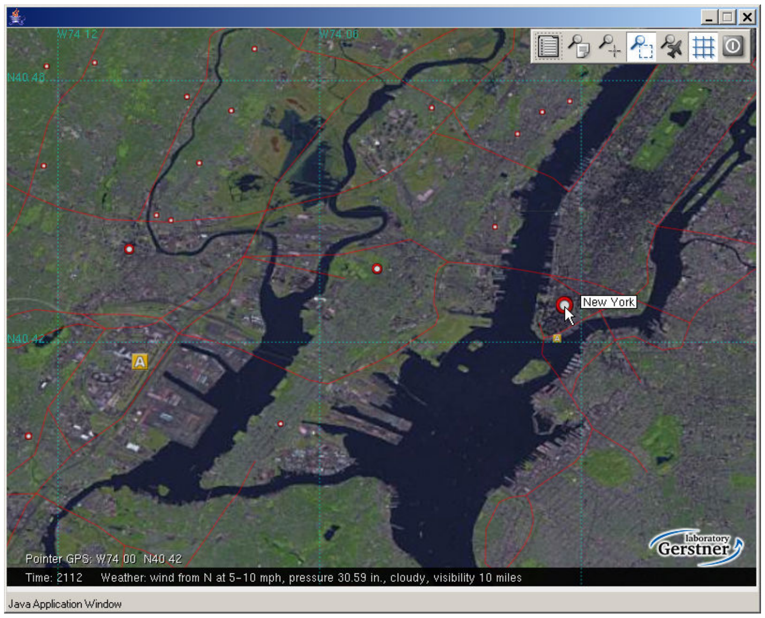
- `arrow keys` – scroll/pan the view,
- `PageUp`, `PageDown` – zoom the view in and out.

As the user changes the view, the appropriate data segment of the virtual map is requested and downloaded from the server, and therefore at any moment the client needs to keep only a minimal amount of data, which results in its fast and efficient performance.

10.1 Internal Data Layers

Apart from the integrated GIS data layers from the external data sources (listed in the Section 8), there are also several internal AGENTFLY data layers that are generated and updated by the simulation system in runtime:

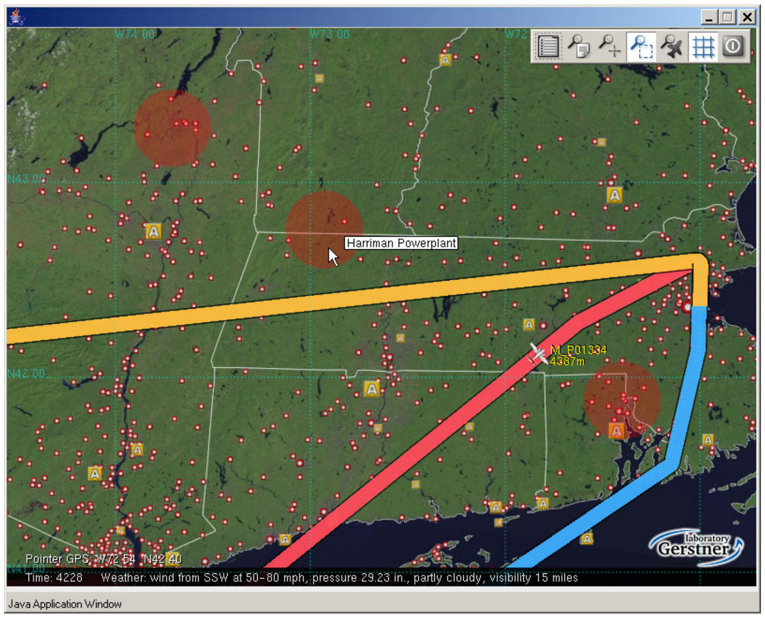
Airplanes – a layer of airplane icons represents each airplane's type, current position and direction. See the Figures 23 and 25.



```
Pointer GPS: W74 00  N40 42
Time: 2112     Weather: wind from N at 5-10 mph, pressure 30.59 in., cloudy, visibility 10 miles
Java Application Window
```

Figure 24: Web access component user view including the weather information

- **Rough Flight Plans** – also called way-points (or mission specification), these plans are series of points in space through which the airplane is supposed to fly on its way. These are displayed as dotted poly-lines in the screen. See the Figure 23.
- **Detailed Flight Plans** – these plans are the actual routes that the airplanes follow. These are displayed as solid poly-lines in the screen. See the Figures 23 and 25.
- **Weather Information** – weather conditions in the local area are displayed in the status line at the bottom of the screen. See the Figures 24 and 25.



Time: 4228 Weather: wind from SSW at 50–80 mph, pressure 29.23 in., partly cloudy, visibility 15 miles

Figure 25: Flight plans avoiding the no-flight zones

11 Visualization Component

The *Operator Agent*, which is an optional component of the AGENTFLY system, allows the human operators to work with the system. This section describes its internal architecture and capabilities for visualization of the system state. Its support for the human-system interface is described in the Section 12. Depending on its current configuration it can provide real-time system state with all important information in a two or three dimensional space to the user.

There can be more operator agents connected to one AGENTFLY system at the same time. Different operators' configurations are shown in the Figure 15. It is possible to connect an operator agent remotely within the intranet network. In such configuration all necessary and requested information providing real-time system state are bridged via the *Topic Bridge Services.* They allow to transmit and share requested data between the system and a number of operators. Each connected operator can be configured using different operator configuration depending on the user access level of an authenticated user. The operator agent can connect to and disconnect from the AGENTFLY system at any time of the simulation.

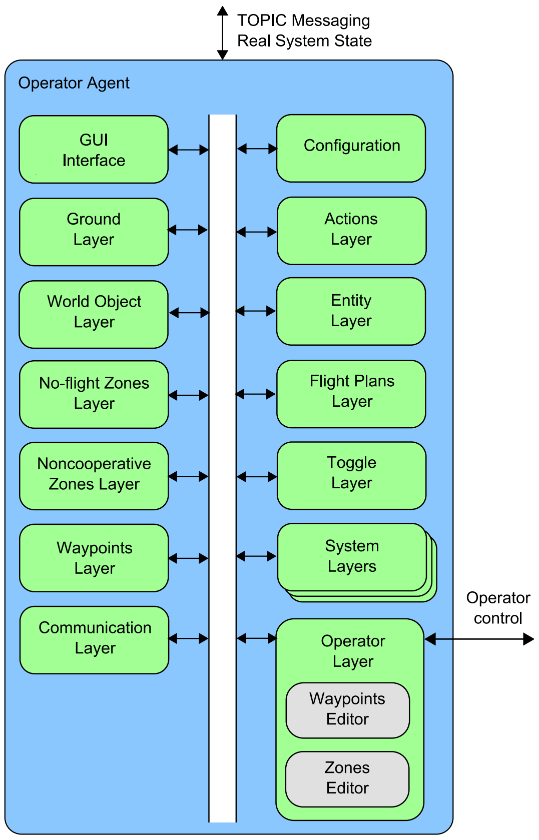


Figure 26: Layer architecture of the visualization component – Operator agent

11.1 Layer Architecture

To enable simple configuration of different functions provided by the operator agent, it utilizes the concept of layer architecture, see Figure 26. The main part of the agent is the *GUI Interface* module providing access to the graphics acceleration via a set of the graphical objects inserted in the scene tree which holds information about rendering camera mode/position and all objects in the scene including their properties.

The *GUI Interface* is built on the top of JAVA-3D framework. This part of the operator agent is responsible for visualization window management. It also parses all window resizing or closing events triggered by the user. Besides the interface for displaying graphics objects in the presentation form to the user in its window the GUI module also provides user input events back to all layers. For example, if a layer wants to receive all mouse events related to the specific graphics object (or a group of objects), it simply registers a mouse listener to the respective scene object.

The camera in the scene graph can be configured to automatically parse users inputs to provide basic functions for changing its settings, such as zoom, scroll/pan/move/change direction the view and switch between its two and three dimensional mode. If the camera object is in focus, the user can use M, Z and Z keys, arrow keys, PgUp/PgDown keys, plus any of the fore-mentioned keys combined with the Shift key, for such actions. The camera pan and zooming in two dimensional mode can be controlled by the mouse too.

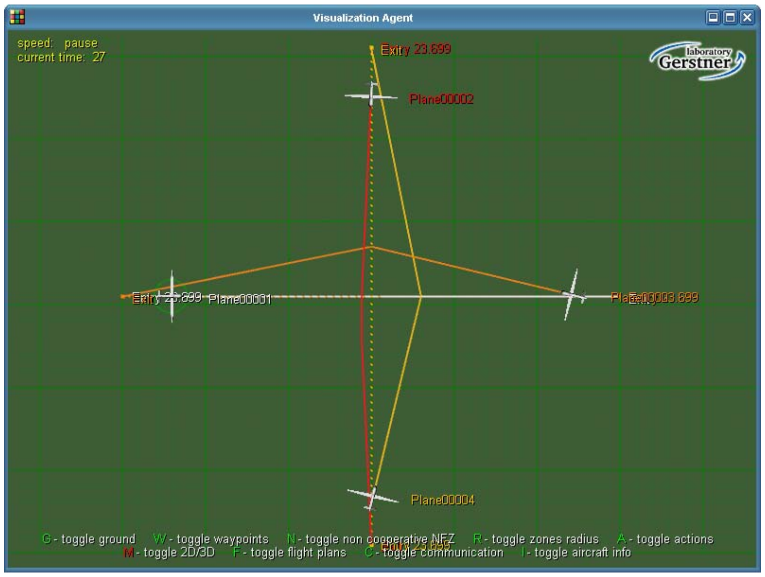


Figure 27: Visualization component in two dimensional mode

The *Configuration* module takes care of the current configuration and starts or finishes respective layer modules. For simple implementation of new layer modules there is a standardized interaction interface among operator agent's components and layers. Each layer module is responsible for subscribing and unsubscribing necessary data from the AGENTFLY system that it would like to present to the user. If a layer module wants to provide more complicated functions to other modules, it can simply do it by its public methods. Each layer module can handle special simulation modes such as the off-line mode, see the Section 3. Within this mode the simulation speed is automatically adapted to the current system load. When the load increases the speed of simulation is decreased and vice versa. This mechanism allows to simulate specific scenario as fast as possible without overloading the system and thus skipping critical situation. In such mode it is not necessary to display all system state in real-time.

There is a set of special *system layers* providing various functionality, for illustration of their visualization form see Figures 27 and 28:

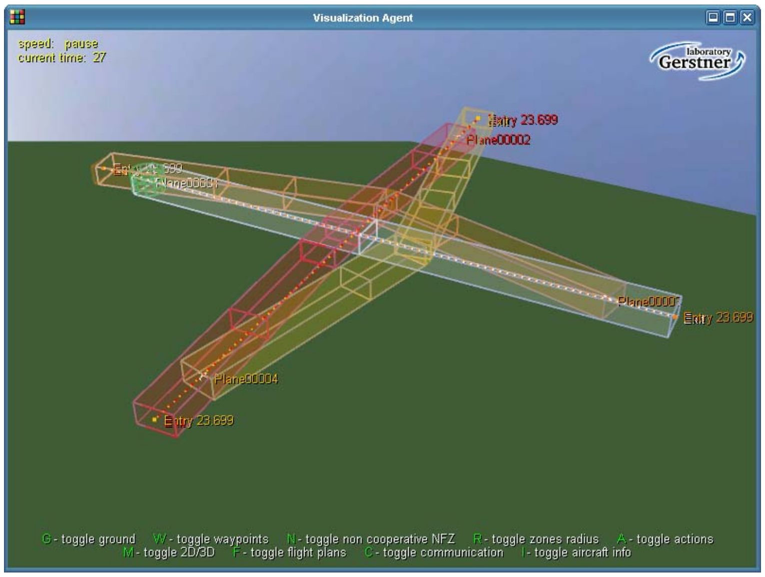


Figure 28: Visualization component in three dimensional mode

- **Time** – displays the current simulation time. It is shown at the left top corner of the window.
- **Current simulation speed** – displays the current simulation speed, but can also parse user input to change the simulation speed. By default, by pressing the number pad plus or minus keys, the simulation speed is increased or decreased. Alternatively, by using the number pad star key the simulation can be paused or start again. The current simulation speed is displayed above the current simulation time.
- **Toggles** – displays the visualization state mode for the registered layers at the bottom of the window. Each layer module can register one or more toggles with a defined number of visualization states to the toggle layer. When the user presses an assigned short cut, the toggle mode is changed and the respective toggle listener is notified about the mode change. This way the user can simply enable or disable visualization of particular graphical layers. It is fully in the responsibility of such module whether it will unsubscribe data which are not displayed at the moment.
- **Coordinates** – shows the current pointer position in the world system coordinates with pre-configured units. The pointer position is shown only if the visualization is now in the two dimensional mode and the pointer is within the bounds.
- **FPS** – is used for the debug purposes to check that all necessary graphical components works properly. The graphics screen is redrawn only if something is changed. To reduce maximal load there is specified maximum allowed FPS for the visualization. E.g. if the simulation is paused and no user control is done, the visualization is redrawn only once per second to update visualized simulation time value.

The rest of the implemented modules can be categorized into two groups. The first group presents system state information to the user and is used only for the visualization purposes, described in the Section 11.2. The second group provides interface for the human operator mode, described in the Section 12.

11.2 Presentation Layers

Ground Layer displays the ground surface in the simulated world to the user. It can show simple flat grounds with a grid or a complex one, e.g. the mountainous area used in the operator scenario (see the Figure 29). The example shows a large area with many details. The ground is divided into many blocks and each block has a defined number of levels of details similarly to the description mentioned in the *entity layer*.

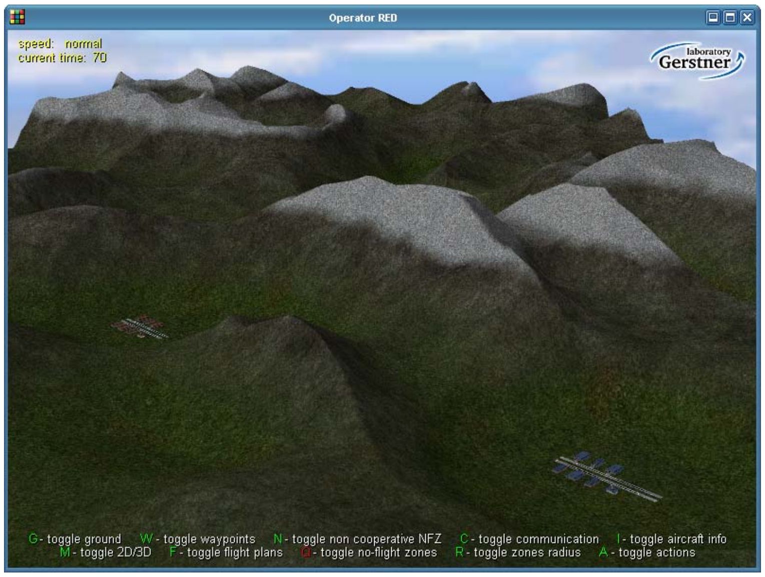


Figure 29: Ground layer displaying simulated mountainous world

World Objects Layer can display any static object in the simulated world. Some of them can be seen in the Figure 29. It has its own configuration with information specifying the object models and their location in the world.

Skybox Layer is able to show any background textures behind the defined world in the visualization. The skybox is constructed as a standard large cube, thus six textures are mapped to each of its side.

No-flight Zones Layer gives information about no-flight zones defined in the system. A no-flight zone encapsulates the airspace where flying is forbidden. In the two dimensional mode top view it is displayed as an orange circle (or another respective shape) and in the three dimensional mode it is shown as semitransparent orange cylinder (or another respective object) in the scene.

Entity Layer is an important presentation layer that shows position and orientation of all simulated or imported aircraft. As the visualization is required to cope with as many as hundreds of airplanes, massive optimization of the visualization especially in three dimensional mode is used. One of the key features is the incorporation of multiple levels of detail (LOD) for all 3D models (Figure 30). For closeup views, the 3D models contain as many details as possible in order to look realistic. But as the camera moves further away, the detailed models are replaced with rougher and therefore more lightweight versions. For wide shots, the 3D models consist only of a few polygons and thus they can be rendered very fast. Text description can be displayed next to each visualized aircraft (Figure 27). The *entity layer* module also provides a set of interfaces that other modules can use to register creation, coloring and selection listeners with specific entities.

The subscriber is simply notified when the user performs a specific action over the respective scene object representing such entity.

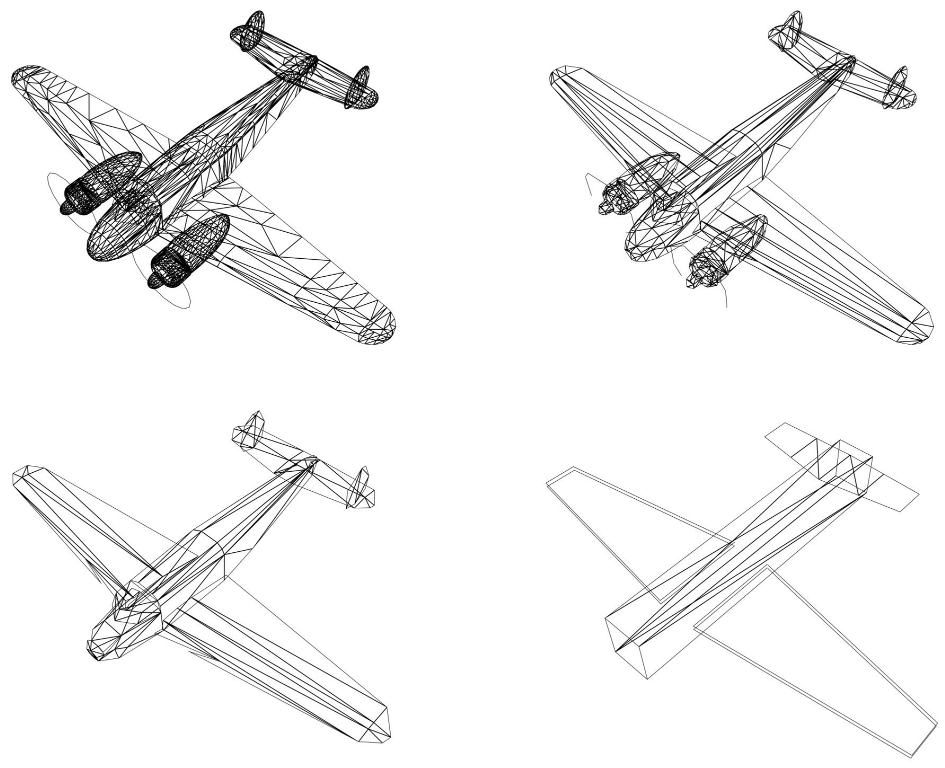


Figure 30: Multiple levels of detail of the same 3D model

Actions Layer presents actions taken by the entity in the form of an icon floating above the model. This form of presentation helps the user to understand what actions are performed by the entity at any moment. Such actions are defined for all implemented collision avoidance algorithms as well as a collision, mission way-point fulfillment, flight plan replanning and many other events.

Flight Plans Layer uses the selection interface of the entity layer to display the current flight plan which is executed by the selected UAA. Flight plans executed by all other UAAs visible on the selected aircraft's on-board radar are also visualized. In the two dimensional view, the flight plans are displayed as the colored solid poly-lines that interpolate through the intended route (Figure 27). In the three dimensional mode, flight plans are visualized as semitransparent corridors of a rectangular profile (Figure 28). Using the coloring interface of the entity layer the flight plan can have the same color as the text description of the UAA.

Waypoints Layer presents the current flight mission of the selected and highlighted aircraft. It is displayed as a set of vertices in the space that are interpolated by dotted poly-lines. There is a text description for each vertex containing its name and time constraints if there are any specified.

Non-cooperative Zones Layer displays all dynamic no-flight zones defined by the currently selected airplane that encapsulate other non-cooperative aircrafts. There can be several zones displayed at the same time as shown in the Figure 31. The zone is displayed as a semitransparent orange object representing the cover border of the dynamic no-flight zone. The shape of the zone can be very complex and therefore it is automatically reduced (approximated) to be rendered quickly.

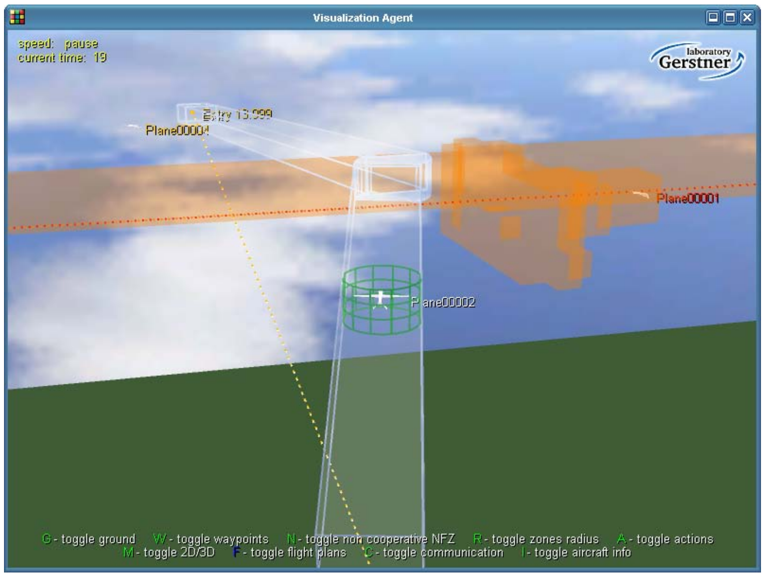

Visualization Agent
speed: pause
current time: 19
laboratory Gerstner
Entry 13.999
Plane00004
Plane00001
Plane00002
G - toggle ground W - toggle waypoints N - toggle non cooperative NFZ R - toggle zones radius A - toggle actions
M - toggle 2D/3D F - toggle flight plans C - toggle communication I - toggle aircraft info

Figure 31: Non-cooperative dynamic no-flight zones around non-cooperative aircrafts

Communication Layer gives a good view of the current negotiation activities among simulated entities to the user. When there is a communication message transmitted between two airplanes (which can occur only in case that they both fall into the visibility range of each another) there is a green line drawn between the displayed objects.

Accessibility Layer provides an info about currently accessible entities. The grey line between entities shows that those two entities can communicate together. If there is no line shown their communication is not allowed.

Universal Visualization Layer is an open layer component used to display various user information different forms to the user without implementing specialized visualization layer. The user uses a local graphical canvas to construct the object for the visualization and this manipulation structure is responsible for its transformation and transmission to all appropriate Operator Agents. If no such data are requested it doesn't do the transformation and only registers the requests for drawing for their later usage. There are provided several graphics objects: box, group, line, linked object, point, color, text and transform graphics. Using linked graphics the user can link the final graphics to the specified entity (e.g. the entity name is always on the right side of the entity). This universal layer is used for many different information provided now in the AGENTFLY, e.g. fuel status, altitude info, predictions from non-cooperative collision avoidance, etc.

12 Human System Interface

The *Operator agent* described in the Section 11 can be configured to provide interface for human operator commands. In such configuration, the user can actively interact with the AGENTFLY system. Currently there are three main functions provided: the user can manage a way-point mission for each aircraft (section 12.1); the user can change the current settings for the *collision avoidance* submodule implementing multi-layer deconfliction mechanisms (section 12.2); and static forbidden no-flight zones management for a group of UAAs (section 12.3).

On the operator side these functions are implemented as the regular layer providers (Figure 26). The *operator layer* utilizes a mouse event listener provided by the *entity layer*. All functions are activated when the user right-clicks on a specific aircraft in the visualization. This works in both visualization modes, two and three dimensional. After receiving such event the operator layer displays local pop-up menu allowing the human operator to select a specific function.

To provide clear way of communication between human operator layer and the pilot agent controlling the UAA, all communication uses agent communication language (ACL) messages sent directly between the layer and the UAA container (see the Figure 14). The Figure 21 presents how the operator control message flow is handled in the UAA container. All messages are tunneled via the *operator bridge* submodule of the plane agent. This architecture allows to replace the plane agent once the pilot agent is deployed to the real UAA hardware. In such case its operator bridge submodule will work as an interface to the radio transmitter connected to the human operator.

As described in the section 9.2.2, the *pilot agent* has modular architecture for the multi-module entity control. If such simulated UAA is able to interact with the human operator, there are *operator modules* loaded that provide end functionality for all provided functions. Depending on the configuration each UAA can provide a different set of functions, similarly to the operator agent.

Each UAA is in one of the following operation states:

- **Parking** – the UAA is not operating at the moment, it is staying on the ground and waiting for the future operation.
- **Take off** – the UAA executes specific flight plan sequence used for taking off from the airport to the full operation mode in the airspace.
- **Full operation** – the UAA is able to fulfill its mission. The mission is defined as a sequence of way-points with specified position. Each way-point can have a time constraint as described in the section 4.
- **Land** – the UAA executes specific flight plan sequence used for landing on the airport.

The *collision avoidance* module of the pilot agent is active only in the *full operation* mode in the airspace. So it means that this module cannot be used for collision avoidance within ground taxiing. Also the collision avoidance during take off and landing phase of the flight must be solved for example by the local airport management. These two collision avoidance cases are not in the scope of the AGENTFLY project now.

12.1 Mission Management

The *mission management* functions enable the human operator to alter the current way-point mission for each UAA. Mission way-points specifying points to be fulfilled by the UAA can be edited only if the pilot agent is in the *parking* or *full operation* mode.

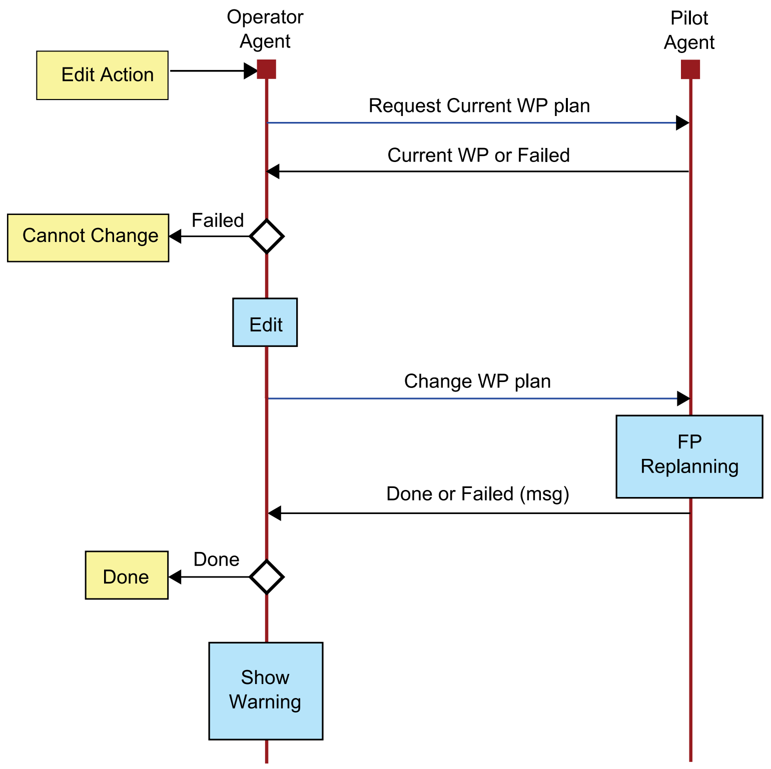


Figure 32: Negotiation between the operator agent and the UAA within the mission edit function

The diagram in the Figure 32 shows the interaction flow in the mission management function:

- When the human operator requests the edit mission function from the context pop-up menu of the UAA, the layer sends a request for the current mission way-point plan to the pilot agent.
- Depending on the state of the UAA the pilot agent responds back with current way-point or a failure message. The pilot agent sends a failure when it is in the take off or landing mode.
- If the operator receives a failure message, it displays a warning that the mission cannot be altered right now. Otherwise it parses the current mission way-point plan and starts frames for the mission editing process.
- At this point the human operator can freely remove any previous way-point or insert a new one. The *operator agent* provides a user friendly interface to manage the mission. When a way-point is selected from the list, the way-point position is automatically highlighted in the world map. When the user wants to insert a new way-point, it can be inserted either directly by its coordinates and altitude or its position can be selected interactively in the map. The user can also select a way-point from the list of predefined points-of-interest. The last way-point for the UAA can specify the return landing airport. If there is no destination airport specified, the UAA automatically adds a way-point of the last airport from which it started to its way-point list. At any moment the user can cancel editing without applying changes.

- When the user applies the changes, the new future way-point list is constructed from the user input and a way-point plan change request is sent to the pilot agent.
- Then the pilot agent tries to make a detailed flight plan from its current position fulfilling the requested way-point sequence. The flight plan re-planning process must still respect the UAA parameters, the world geography and the defined static forbidden no-flight zones. If there is an unreachable way-point, the pilot agent reports an error back to the operator agent. If the re-planning process finishes successfully with no exception thrown, the all-done message is reported back to the operator agent.
- If everything goes well, the mission of the UAA is changed and the mission edit is finalized. If there was a re-planning problem with an unreachable way-point, the problem is reported to the human operator and he can continue to edit the mission and solve possible problems. In such case the interaction protocol resumes its execution at the *edit* action again.

12.2 Switching the Collision Avoidance Method in Real-Time

The operator agent provides special function for *changing the current collision avoidance methods* used by the UAA. The human operator simply selects the desired collision avoidance configuration from the local pop-up menu of the specific UAA. The collision avoidance configuration defines which collision avoidance solvers will be used and how by the multi-layer collision avoidance architecture. The possible configuration is described in the section 13.

Interaction between the operator agent and the UAA is quite simple:

- When the user requests to change current collision avoidance setting, the operator agent sends a request message with the desired new multi-layer collision avoidance configuration to the pilot agent.
- The pilot agent checks if the configuration can be changed at this moment. If so, it simply restarts the collision avoidance module of the pilot agent with the new requested configuration (see Figure 21). Then it sends the all-done response back to the operator agent. Otherwise if the configuration cannot be changed now, it sends a failure message back. The configuration can be changed only when the UAA is in the *full operation* mode due to the fact that its functionality is enabled only in this mode.

12.3 Static No-flight Zones Management

As described in the section 9.2.2, each UAA can have several types of forbidden no-flight zones defined: *world zones*, *static zones* and *dynamic zones*. *Static zones* encapsulate world areas where the UAA cannot operate, e.g. enemy forces area, etc. These zones can be altered by this function of the operator agent.

static zones can be shared by a group of UAAs. In such case the static zones are edited for all respective UAAs at the same time. The static zones for the UAA can be edited only when the aircraft is in the *parking* or *full operation* mode.

The interaction flow in zone management mode is more complicated. Besides the *operator agent* and UAAs' *pilot agents* there is also a *zone manager* entity. This is the entity responsible for no-flight zones management. It is useful mainly in the simulation, because it allows to share

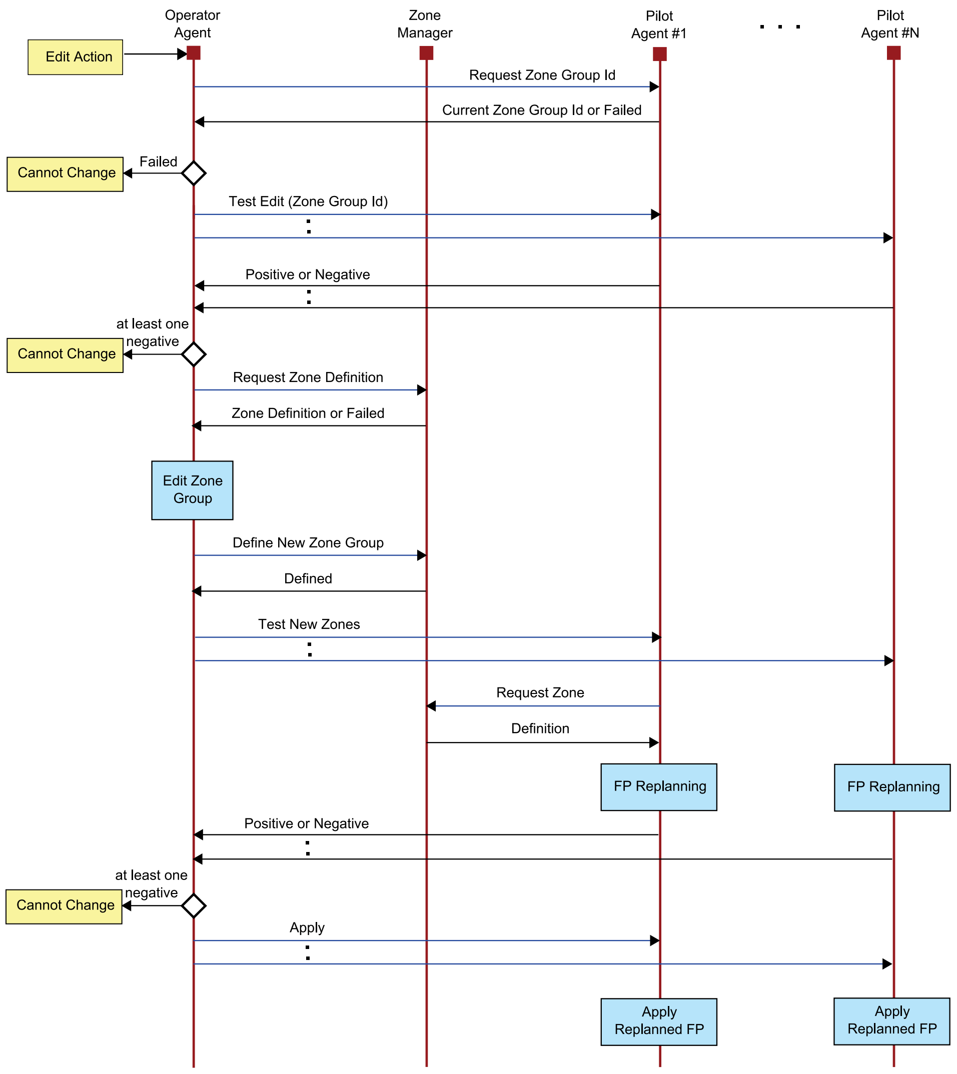


Figure 33: Negotiation protocol used for changing static zones for a group of UAAs

common large octant trees with defined no-flight zones among all entities within a single JVM and thus it allows to save more system resources when more UAAs are simulated on the same host computer. The interaction is shown in the diagram 33:

- When the human operator requests the zone edit function from the local pop-up menu of an UAA, the layer sends the request for the current zone group identification to the respective pilot agent first.
- The operator module of the pilot agent checks if the static zones for the UAA can be altered now. The zones cannot be changed in the *take off* and *landing* mode. It replies back with the current static zones identification. If the zones cannot be changed now, it sends a failure message back.
- If the operator agent receives a failure response, it displays a warning message to the human operator and the operation is canceled. Otherwise, it sends query to all other members of

the UAA group using the same zone group definition. The query is used for testing if all other members can also change the zones now (if they are in appropriate modes). The query is not sent to the pilot agent which has been already asked for the zone group identification. If there is no other member of the group using the same static zones group, no queries are sent.

- All pilot agents receiving the query send positive or negative response depending on their current operation mode.
- The operator agent waits for all responses and if there is at least one negative response, it displays a warning message to the human operator and cancels the operation.
- If the zone group can be altered now for all UAAs using the same group, the request for the current zone definition to the *zone manager* is sent.
- The zone manager sends the zone definition response if the zone group is defined. Otherwise, it sends a failure message back.
- At this moment, the frames for the static forbidden no-flight zones editing process are shown to the human operator. And he is able to alter the current definition. He can freely remove the already defined one or define a new one. The *operator agent* provides a user friendly interface to manage a group of zones. When a zone is selected in the list, it automatically highlights the zone area in the world map. When the user wants to insert a new zone, it can be inserted directly by specifying the coordinates of its center and radius. Its parameters can also be easily inserted from the map interactively. The user is also able to select the zone from the list of predefined zones. At any moment the user can cancel the editing process without applying the changes.
- When the user applies the changes, the request for definition of a new zone group is sent to the *zone manager*.
- It responses back when new zone group is successfully defined.
- Then a test query is sent to all related pilot agents with this changing operation. The query asks each pilot agent if it can accept the changed static zones. The zones can be accepted if there is no current way-point in its mission that is not satisfiable after the update, but before it was.
- To perform such a check, the pilot agent tests if the changed zone definition is available in its JVM first. If it is not known yet, it simply sends a request for zone group definition to its known *zone manager* entity and waits for the definition. Then it makes the definition available to all other entities within the same JVM.
- At this point, the pilot agent has all necessary information to perform the query test. It simply starts its flight plan re-planning process from the current position of the UAA. If a new detailed flight plan cannot be found due to any reason respecting the new static no-flight zone definition, the pilot agent sends a negative response to the operator agent's test query. Otherwise, it sends a positive response.
- The operator agent waits till it receives all responses from all UAAs using the same edited static no-flight zone group. And then, it tests if there is at least one negative response. If such response exists, the agent displays a warning message to the human operator saying that the changed zones cannot be applied now due to the specific reason. Then the editing process in the operator goes back to the *edit zone group* point where the user can adjust its modification again.

- Otherwise, if there is no negative response to the test query, the *operator agent* sends a notification saying that the zone definition is applied to all UAA members using the same group. And the edit zone group operation is finished from the operator agent side.
- When the pilot agent receives such notification, it simply takes the already re-planned version of its flight plan which it has from the test after queering. It applies this new flight plan as the current one for execution.

13 Multi-layer Collision Avoidance Architecture

We have designed several collision avoidance methods that can be used for collision avoidance of the autonomous airplanes. In the dynamic environments that requires frequent mission replanning and in environments with high number of non-cooperative objects, multiple collision avoidance methods need to be used simultaneously.

The multi-layer *collision avoidance module* of the pilot agent (see the Section 9.2.2) solves the future collisions using the combination of different collision avoidance methods. The collision avoidance module works in the fully distributed manner and doesn't utilize any central planner for the collision avoidance of physical entities. The module architecture is domain independent. Therefore it is ready for the deployment on the autonomous vehicles like airplanes, robots, cars, submarines, etc.

The next section presents the basic architecture of the collision solver manager. The Section 13.2 describes its configuration in detail and configuration examples are stated in the Section 13.3.

13.1 Collision Solver Manager

Collision solver manager (CSM) is the general module used for the solving of future collisions between autonomous entities. It is implemented as a plug-in module that can be enabled in every agent. This module is domain independent and can be used from any agent that offers the necessary interface needed for the interaction. Its integration to the AGENTFLY to the pilot agent is described in the Section 9.2.2. The interface between the agent and the plug-in defines following functions:

- creation/finishing CSM module,
- method for change of the agent's current plan,
- method that notifies the CSM about agent's plan change,
- methods for (de)registering/using agent's radar,
- methods for (de)registering/using agent's idle task; this means, that module can receive messages with specified protocol/performative that are unique,
- methods for (de)registering/using agent's time,
- methods for getting address of other agents (from radar data),
- method to get agent's position,
- method to get knowledge base,
- some other less important methods.

Parameters used in these interfaces are also generalized. The most important parameter is the agent's plan interface that is implemented in each domain in different way. So the CSM can be used as well with various entities providing that planning interface such as robots, cars, ships, submarines etc.

The main function of the CSM is to make the decision which collision will be solved as well as the decision which solver solves the collision. It is selected from the collisions registered by collision solvers. The collision solver (CS) is a module that is able to detect and solve future collision. It can be general in the same way as the CSM or it can by domain dependent – cooperative collision avoidance solvers can be domain independent but non-cooperative collision avoidance solvers depend on the selected domain.

In the typical case there are several registered (or one) collision solvers to the CSM. The CS notifies the CSM about collision when it is detected. More CSs can notify about the same collision and one CS can notify about more than one collision. The CSM collects all notifications and selects which one is the most important to be solved. The selected solver gets specified amount of processing time to solve selected collision. Other collision modules are waiting until they will be selected or their collision doesn't exist anymore.

Solving the collision in CS can take a long time. For this reason there is a computational thread in the CSM and the CS part responsible for the collision solving is started in that special thread. As a result, the CSM can control the length of the time assigned to the CS (solving timeout) and process the communication and sensors necessary for the flight during the computation. When this timeout is exceeded for the selected collision solver, the computational thread is interrupted and a new collision to be solved is selected.

The CSM is notified when an arbitrary CS detects any future collision. This notification contains the identifier of the collision object (aircraft), the reference to the solver, the time of the collision and solver's data for its private purposes. Only the earliest time is saved for each collision (identified by that ID). The notification received from the cooperative CS about collision with non-cooperative aircraft is discarded. The CSM selects a collision to be solved using the following rules:

- Find the first valid collisions. The first collisions are the earliest ones. Times of the collisions are taken as the same if the difference between them is less than `SAME_COLLISION_INTERVAL_LENGTH` constant. The first collisions are those with the earliest collision times using this tolerance. The valid collision is a collision registered by the collision solver that can be selected for solving (i.e. still appears).
- If there is no collision being solved at the moment, the collision with the highest priority is selected from the set of first valid collisions (as specified in the last item).
- If the time of the earliest valid collision is earlier than the time of the collision which is currently being solved, solving of the current one is interrupted and the collision with the highest priority is selected from the set of first valid collisions.
- If the time of the earliest valid collision is the same as the time of the collision that is being solved, the priority is compared to the the highest priority from the set of first valid collisions. If the current collision has higher priority, nothing happens. Otherwise solving of the current collision is interrupted and the collision with the highest priority is selected from the set of first valid collisions.
- If the time of the earliest valid collision is later than the time of the current collision, nothing happens.

The CSM selects appropriate collision solver to solve the selected collision (given by the rules above) using cooperative or noncooperative ordered list of solvers. The difference is counted between the time of the future collision and the current time. The difference determines which collision solver will be used (corresponding time interval in the list is selected). If the selected CS is a blank solver, it is selected the next CS to the blank solver (closer to future collision).

The CSM counts the maximal time that can be used by selected collision solver to solve the earliest collision. This time is the minimum of the *MaxSolvingTime* parameter of the CS and the rest of the time interval assigned to the selected CS on the defining time axis (see the Section 13.2). If the time for selected solver is shorter than *MinSolvingTime* parameter of the CS, the next not a blank collision solver is selected. The counted time is used as an execution timeout. When timeout elapses, the solving of the collision is terminated.

Current flight plan is set as the fixed (unchangeable) to a *change point* that is equal to the timeout assigned to the collision solver. This prevents modification of the flight plan in the history as defined in the Section 4.2.

13.2 Configuration Description

The collision solver manager is configured by its own configuration xml file that contains a list of collision solvers. Each solver has following parameters specified:

Name - name of the solver.

ClassName - name of the solver's Java class. This class is dynamically loaded using Java class reflection. If the name is *blank*, no collision solver is used and only empty time interval is inserted.

Cooperative - determines if this solver can be used for cooperative collision avoidance.

UseInterval - defines the length of the time frame when the CS can be used for solving particular collision. The zero time represents the unbounded time length and such setting can be used only for the first CS in the list.

MaxSolvingTime - the maximum processing time that can be assigned to the CS.

MinSolvingTime - the minimum time that the CS needs to solve a collision.

SolverConfigurationConfObjectName (optional) - defines the configuration for the collision solver.

The CSM preprocesses its configuration during the initialization. There are created two ordered solver lists - cooperative and noncooperative. The cooperative list contains all collision solvers defined in the configuration file in the same order as in the list. The non-cooperative solver list contains only solvers where the *cooperative* parameter is set to *false*.

The solver list represents time axis (see the examples in the Section 13.3), where each collision solver occupies defined part of the axis. The time axis begins at the current time and goes to the time of the future collision. The collision solvers are ordered in the same sequence as in the configuration list. The list is filled from the time of the future collision backwards using the *UseInterval* parameter as a length of the interval. The rest of the time axis is filled by the first collision number (see Figures 34 and 35 in the examples). The priority is set to each collision solver according to its position in the ordered solver list. All collision solvers are started during the initialization and its collision detection parts run continually.

13.3 Configuration Examples

This section provides an example configuration of the CSM and its application to the selected cases. The example is only illustrative to show how CSM handle different situations. The typical configuration will contain cooperative solvers at the beginning, noncooperative solvers in the middle and blank solver at the end.

```
<Solver Name="Solver1"
        Cooperative="true"
        ClassName="BestandSlowCooperativeSolver"
        UseInterval="0"
        MaxSolvingTime="10000"
        MinSolvingTime="5000"
        SolverConfigurationConfObjectName="BASCOOP_CONF"/>
<Solver Name="Solver2"
        Cooperative="false"
        ClassName="NoncooperativeSolver"
        UseInterval="5000"
        MaxSolvingTime="3000"
        MinSolvingTime="1000"
        SolverConfigurationConfObjectName="NONCOOP1_CONF"/>
<Solver Name="Solver3"
        Cooperative="true"
        ClassName="BasicCooperativeSolver"
        UseInterval="10000"
        MaxSolvingTime="4000"
        MinSolvingTime="2000"/>
<Solver Name="Blank"
        Cooperative="true"
        ClassName="blank"
        UseInterval="7000"
        MaxSolvingTime="0"
        MinSolvingTime="0"/>
<Solver Name="Solver4"
        Cooperative="false"
        ClassName="NoncooperativeSolver"
        UseInterval="10000"
        MaxSolvingTime="2000"
        MinSolvingTime="2000"
        SolverConfigurationConfObjectName="NONCOOP2_CONF"/>
<Solver Name="Blank"
        Cooperative="false"
        ClassName="blank"
        UseInterval="5000"
        MaxSolvingTime="0"
        MinSolvingTime="0"/>
```

- The Solver1 has the *UseInterval* set to zero. This parameter has no sense for the first CS in the list.
- The Solver4 has *MinSolvingTime* and *MaxSolvingTime* set to the same value. This means that *MinSolvingTime* will be always assigned to it exactly to solve the collision.

- The names of the solvers have to be unique except the blank solvers.
- Solvers can use the same Java classes (e.g. Solver2 and Solver4 uses the same). Different configurations may be used for each solver.
- The *SolverConfigurationConfObjectName* is not mandatory parameter.

The Figure 34 shows the cooperative pattern generated from the configuration above. It contains all solvers in the same order. Solver1 has the highest priority.

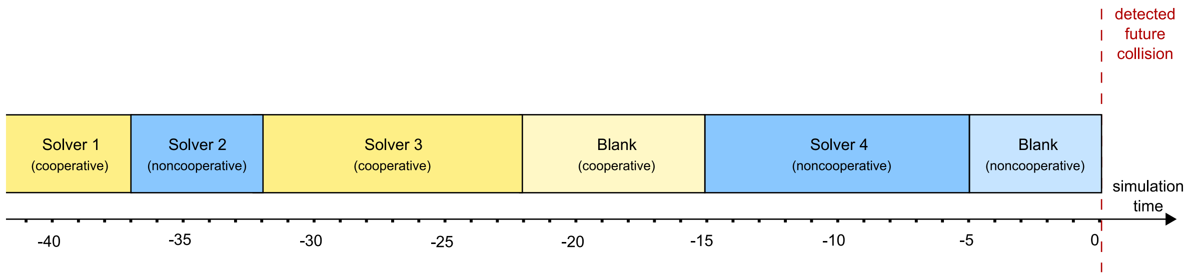


Figure 34: The cooperative pattern of the configuration related to the time axis.

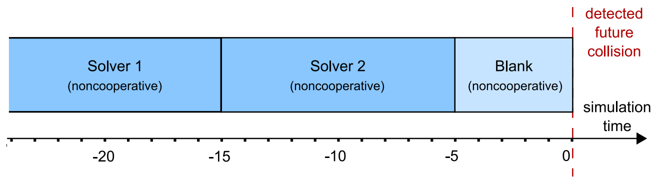

detected future collision
Solver 1 (noncooperative)
Solver 2 (noncooperative)
Blank (noncooperative)
simulation time
-20
-15
-10
-5
0

Figure 35: The non-cooperative pattern of the configuration related to the time axis

The Figure 35 shows noncooperative pattern. It contains only non-cooperative solvers in the same order as in the configuration excluding cooperative solvers.

Example 1: 25 seconds to the collision

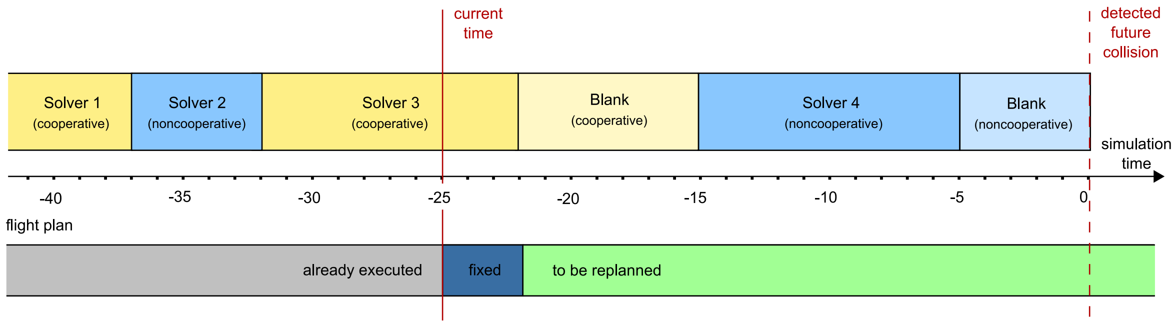


Figure 36: 25 seconds to the collision

The Figure 36 shows the situation when there is 25 seconds left to the collision. The *solver3* is selected according to the list of the solvers. The selected timeout for solving is the rest of the time until the end of the solver's time interval on the axis. This means that the *solver3* can use computational thread of the CSM for next 3 seconds. If no solution is found, the CSM interrupts its execution and starts another solver.

Example 2: 12 seconds to the collision

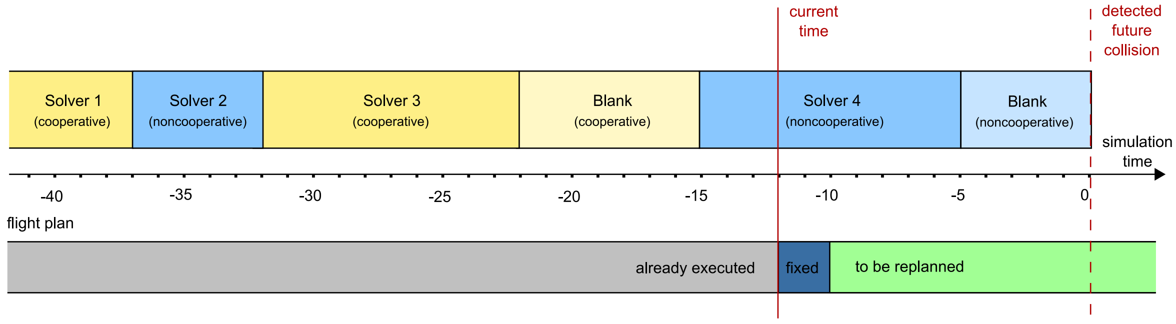


Figure 37: 12 seconds to the collision

The Figure 37 shows the situation when there is 12 seconds left to the collision. The *solver4* is selected. The selected timeout is the minimum from the rest of the time until the end of the solver's time interval and *MaxSolvingTime* parameter. The *solver4* has set this parameter to two seconds. Thus selected timeout will be 2 seconds. Therefore the solver has short fixed part of the plan and can react very fast. If no solution is found, the same solver can be selected (started) again after 2 seconds until its time interval in the list of solvers exceeds.

Example 3: 18 seconds left to the collision

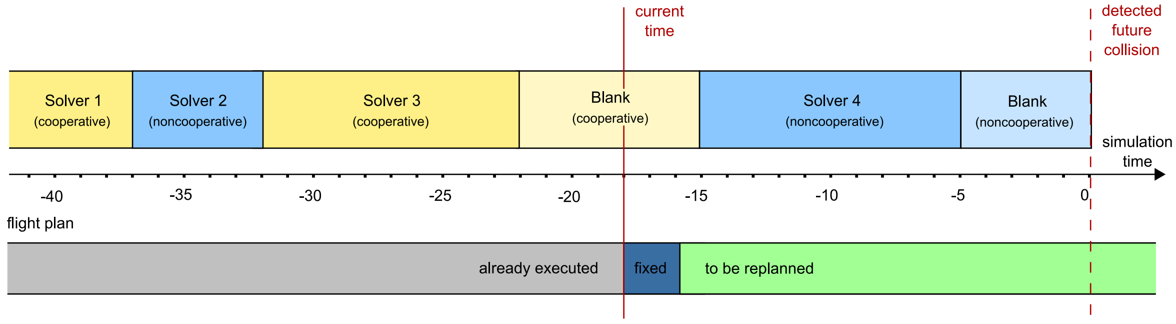


Figure 38: 18 seconds left to the collision

The Figure 38 shows the situation when there is 18 seconds left to the collision and the blank solver is used. The next solver will be selected - *solver4*. The timeout is the minimum from *MaxSolvingTime* (2 seconds) and the rest of solver4's interval (13 seconds).

Example 4: 23 seconds to the collision

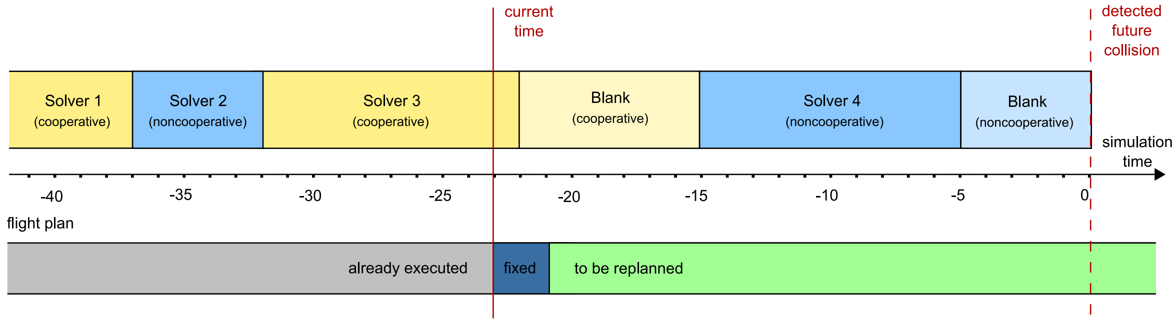


Figure 39: 23 seconds left to the collision

The Figure 39 shows the situation when there is 23 seconds left to the collision and the *solver3* is used. The timeout assigned to the solver3 should be 1 second. This is less than the *MinSolvingTime* parameter of the solver3 (2 seconds). The next non-blank solver is selected - *solver4* with the 2 seconds timeout. Any collision solver cannot be selected for execution after the end of its interval minus *MinSolvingTime* parameter anymore.

Example 5: 3 seconds left to the collision

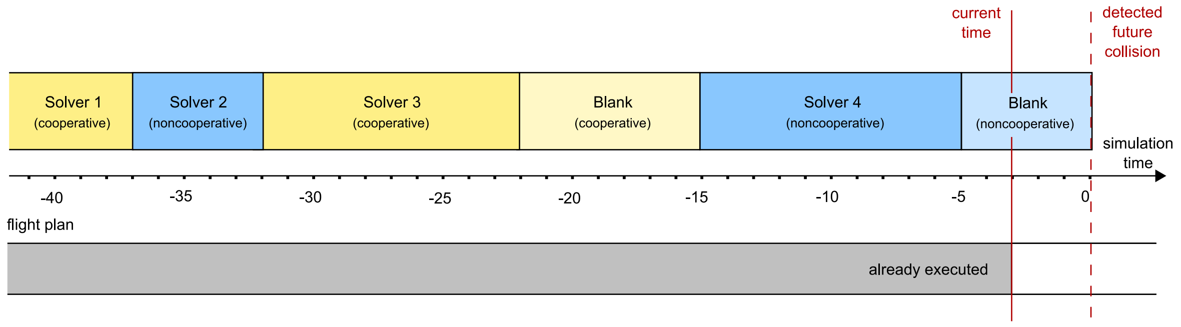


Figure 40: 3 seconds left to the collision

The Figure 40 shows the situation when there are 3 seconds remaining to the collision and the blank solver is used. There is no solver behind this blank solver, so the CSM will not select any solver. The same situation can arise up to 7 seconds before the collision (5 seconds for the blank solver and 2 seconds is the *MinSolvingTime* parameter for the last solver).

14 Cooperative Collision Avoidance

All implemented cooperative collision algorithms in AGENTFLY system provide a local collision avoidance, see the definition of the avoidance problem in the Section 4.5. The algorithms are based on the communication and the negotiation among airplanes. Such algorithms can be used also in the domain where airplanes can communicate only with planes located nearby (there is limited communication range). Well known techniques for collision avoidance based on the potential fields are not suitable for the specified domain (3D, no-flight zones, the need of smooth flight plan also in its first derivation, allowed speed changes and the fact that the airplane cannot stop) due to their complexity and because of the complicated dynamic mission airplane specification.

In the AGENTFLY there are implemented three cooperative methods: rule-based (Section 14.4), iterative peer-to-peer (Section 14.5) and multi-party collision avoidance (Section 14.6). All described algorithms are implemented as collision solvers in the multi-layer architecture (Section 13). The 'See' ability of the algorithms is the same, see the Section 14.1. It is implemented as the transponder negotiation task in the AGENTFLY (Section 14.2). The changes of the flight plan provided by the cooperative algorithms are done via the evasion manoeuvres (Section 14.3).

14.1 Local Cooperative Detection

The *see* capability [2] in the AGENTFLY is implemented by negotiation and flight plan comparison. Due to the limited communication range each airplane A_i is aware only of planes located within this range $A_j \in \widetilde{\mathcal{A}_i}$. $\widetilde{\mathcal{A}_i}$ denotes the set of airplanes $A_j \in \mathcal{A}$ of which A_i is aware of. The described algorithms solve encounters locally where they can be detected. It is not necessary to identify collision $A_i \bigotimes A_j$ for whole fp_i and fp_j. The airplane can share only *part of its current flight plan* $\widehat{fp_i}$ from current time t_c for interval t_{share} where $\mathbf{p}(\widehat{fp_i}, t) = \mathbf{p}(fp_i, t)$ for $\forall t : t_c \leq t \leq t_c + \delta_t$. δ_t is selected by the airplane not to expose all its future plan including its mission objectives, but to give enough part to identify the possible collision. Such local sharing of the flight plans also reduces the necessary communication flow. The flight plan sharing is implemented by the subscribe-advertise protocol [14]. Every time when airplane is aware of new other airplane A_j it subscribes for its $\widehat{fp_j}$. The plane A_j, by accepting subscription request from A_i, will provide regular updates of its $\widehat{fp_j}$ such there will be enough part of future part of the flight plan from current time. If A_j changes its fp_j for any reason – change of its mission objectives or as a result of other collision avoidance – it provides new fresh $\widehat{fp_j}$ to the subscriber as soon as possible.

Airplane A_i who received $\widehat{fp_j}$ from all its neighbors $\widetilde{\mathcal{A}_i}$ *performs check* if $\exists t$ where $\mathbf{col}(fp_i, \widehat{fp_j}, t) = 1$ upon every received update. If such t exists A_i tries to identify the first and the last collision point $t_1^{A_i \otimes A_j}$ and $t_2^{A_i \otimes A_j}$ (see the Figure 41). Airplanes are also able to detect multi-collision group $\mathcal{A}_\mathcal{C}$ by exchanging information about collisions. A_i prepares its local view of an encounter $en_k^{A_i} = \langle t, \{fp_i\} \bigcup \{\widehat{fp_j}\}_{A_j \in \mathcal{A}_\mathcal{C} \setminus A_i} \rangle$. Selection of $t_1^\mathcal{C} > t > t_c$ depends on the used algorithm or combination of algorithms and is chosen somewhere between current time t_c and time of the earliest collision $t_1^\mathcal{C}$ for the given multi-collision $\mathcal{C}$. $t - t_c$ defines the timeout which is then given for invoked collision avoidance algorithm. If the result for $\mathcal{C}$ is not provided within the specified timeout, the algorithm is interrupted and next iteration is invoked for the same $\mathcal{C}$. Note that when the local cooperative detection the encounter contains full fp_i and only parts of $\widehat{fp_j}$, but it is enough to do distributed local collision avoidance. The LCAP algorithm still provides solution containing full flight plans fp_j for $\forall A_j : A_j \in \mathcal{A}_\mathcal{C}$ because all flight plans are still provided by its final implementor A_j.

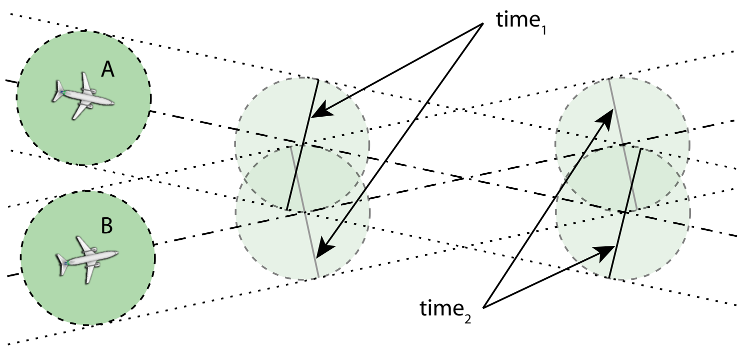


Figure 41: Identification of the first and the last collision point between current flight plans of airplanes A and B.

14.2 Transponder Negotiation Task

The transponder task is used for general communication between the autonomous agents representing aircrafts in the system. The main feature of the $\mathcal{A}$-**globe** task [13] is that all incoming communication from the appropriate opposite task is automatically routed and handled by the task. The implemented cooperative collision avoidance algorithms use the peer-to-peer negotiation. The task presented in the Figure 42 is used for each pair of negotiating airplanes in the system. The transponder task is described from the view of the airplane A, but the second airplane does the same from its point of view. Brief description of the transponder task is below.

Lets suppose an airplane A flying along its planned optimal flight path fulfilling its mission way-points with their time constrains (described in the Sections 5 and 6). The airplane B enters the alert zone (radar range) of airplane A. The pilot agent of the airplane A is notified about its position and the flight code by the on-board radar system, see Figure 43 left. The pilot agent checks if there already exists transponder task with airplane B. If there is not such a task, it creates the new one and tries to establish negotiation connection with the pilot agent B by sending *initiate message*. This message contains information about the plane type and information if pilot agent A has airplane B on its own on-board radar system. If the connection cannot be established or the communication is not trusted, the pilot agent should use *non-cooperative collision avoidance* method against the plane notified by the on-board radar, described in the Section 15. Each transponder task holds information about mutual radar status of involved airplanes. If airplane B is on the radar of airplane A, true value in the variable *onMyRadar* in the appropriate task of plane A is stored. If airplane A is on the radar of airplane B, true value in the variable *onOtherRadar* in the same task is stored. The opposite airplane does the same in its own task from its point of view.

The transponder task of pilot agent A is in the *initiating state* and is waiting for *initiate message* from pilot agent B. When pilot agent B receives *initiate message* from pilot agent A (and has no transponder task with pilot agent A), it creates new transponder task to communicate with airplane A and sends back *initiate message*. At this moment the transponder task of the pilot agent B is initiated (*working state*) and sends flight plan future part for the specified amount of time depending on the flight speed. When pilot agent A receives the *initiate message* from the pilot agent B, it is initiated as well (changes state to *working state*) and sends its flight plan future part to the pilot agent B.

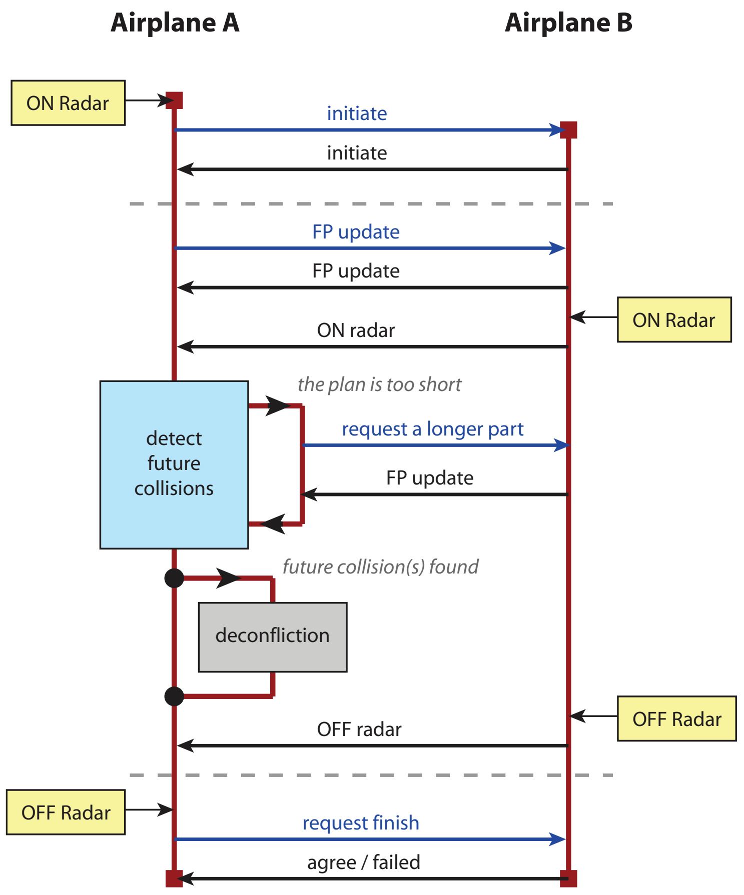


Figure 42: The transponder task message flow overview.

The transponder task communication flow is protected by the system of timeouts. If transponder task A does not receive *initiate message* in the specified amount of time, it re-sends its *initiate message* again.

It can happen, that *initiate message* is sent by the pilot agent A, but before it is received by the pilot agent B, on-board radar system of airplane B detects the airplane A and so the pilot agent B sends its own *initiate message* to the pilot agent A. It is possible due to fact, that *initiate messages* are symmetric and both of them will serve as answers as well. So both transponder tasks will be initiated, when appropriate initiate message will be delivered, see the Figure 43 right.

At this moment both transponder tasks are initiated and in the *working state*. Both sides of the transponder task can send these messages in this state:

Flight plan message - it is sent every time (i) due to change of its own flight plan, (ii) after the predefined regular time period expires or (iii) as an answer for the special request from the other side of the transponder task.

Radar change message - sent when the change occurs in its on-board radar system (other airplane appear/disappear from the radar).

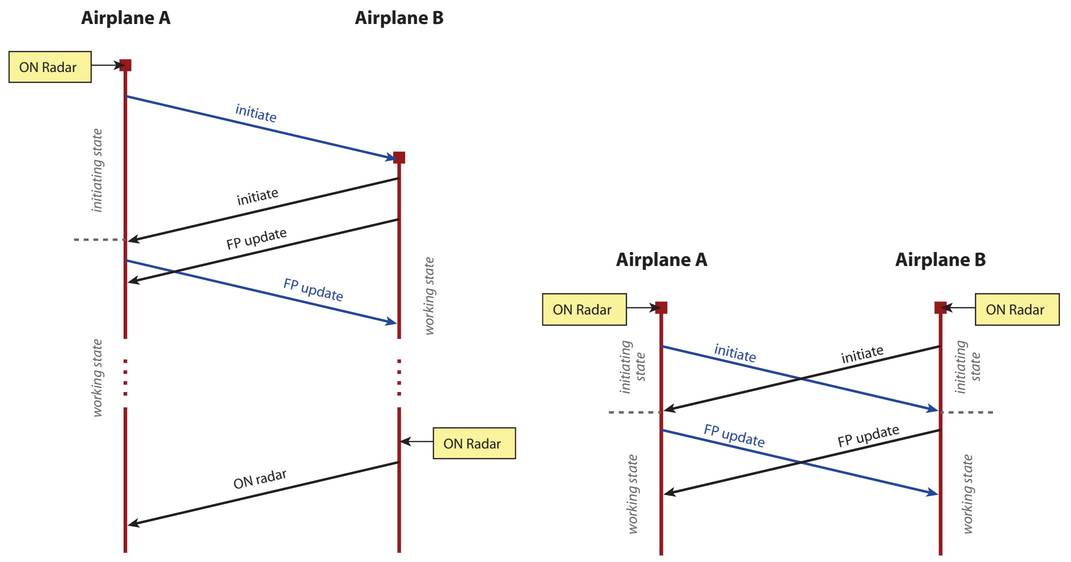


Figure 43: Two cases of the initiating sequence of the transponder task: the A detects B first (left), both A and B detects the other at the same time (right)

Request for the longer plan message - used when the pilot agent needs more information about the flight plan of other airplane because of its collision detection method (see the Section 14.1).

These messages can be sent in any order due to the needs of each pilot agent. Every time when the task receives the *flight plan update* from the opposite plane it executes the *collision detection*, see the Section 14.1. The collision detection process needs to identify the first and the last collision point between two flight plans. So, in the case that airplanes exchange only local parts of the flight plans it is possible that the last collision point cannot be identified without having longer part of the other flight plan. For this reason there is implemented the special request with a longer part message with the flight plan update response communication, see Figure 42. If a collision between the current flight plan and received one is found the collision avoidance process is started (grey block in the figure). This is done by inherited collision solvers, where specific algorithms are implemented. Currently one of the following methods is used: (i) rule-based (described in the Section 14.4) or (ii) utility-based (see the Section 14.5).

When some unexpected events occur, it is possible that transponder task receives the *not-understood message*. As a reaction, it sends the *restart message* and new *initiate message*. When other side of the transponder task receives the *restart message*, the set its state to *restarting state* and waits for *initiate message*. The rest of the communication is the same like described above within the standard initiation sequence.

When airplane (for example B) leaves the alert zone (radar range) of the airplane A, the pilot agent A is notified about this by its on-board radar system. If the pilot agent A has the information in the transponder task that airplane A is not on its radar, see Figure 44 left, the task starts negotiation sequence to finish the task. Its state is changed to the *finishing state* and the *request finish message* is sent. The transponder task does not react to any reaction to any message (except messages related to finish process), but it remembers them to use them later if needed.

When the transponder task B receives *request finish message*, it changes the variable *onOther-*

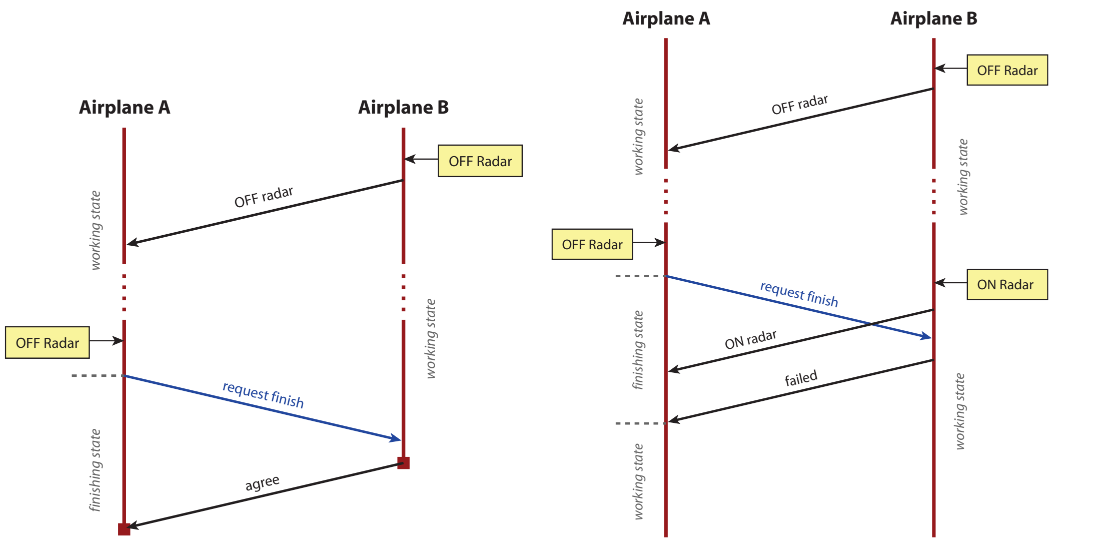


Figure 44: Left: the standard finishing sequence of the transponder task. Right: the interruption of the finishing sequence due to the received visible notification from the on-board radar.

Radar to false. When both variables *onMyRadar* and *onOtherRadar* are false, the transponder task B sends back *finish agree message* and transponder task is removed from pilot agent B. When the variable *onMyRadar* is true, transponder task B sends back *finish fail message* and normally continues in work, Figure 44 right. This situation can be caused by the distributed nature of the negotiation. For example the transponder task A had sent the *request finish message* before it received the *radar change message* from transponder task B.

Transponder task A can receive three messages as an answer to its *request finish message* in the specified amount of time (before the timeout exceeds):

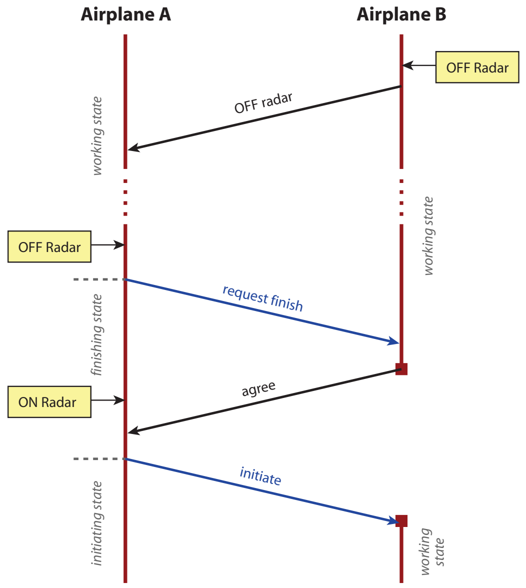
finish agree message - the A checks again if both variables *onMyRadar* and *onOtherRadar* are false. If so the transponder task A is removed from the pilot agent A and the entire communication between A and B is closed, situation in the Figure 44 left. If the variables are not false, restart of communication is processed using the *restart message* as described in the previous part, Figure 45 left.

finish fail message - the task A changes its state to *working state* and processes all messages, that have been received and remembered during *finishing state* as described above, Figure 44 right.

request finish message - this message can be received in similar situation like in initiating of the transponder task. It can happen that transponder task B starts its finishing at the same time and sends the *request finish message* before receiving the *request finish message* from the transponder task A, Figure 45 right. The reaction to this message is the same like to the *finish agree message.*

When the timeout exceeds and no message is received, transponder task A repeats sending of *request finish message.*

If airplane A detects the plane B on its on-board radar system during the finishing state, the transponder communication has to be initiated again. The transponder task B can be already



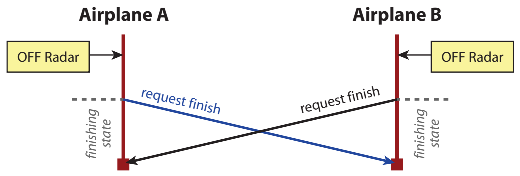


Figure 45: Left: the communication is re-initiated due to the changing the radar state within the finishing sequence. Right: invocation of the finishing state by both tasks at the same time.

removed, so the transponder task A has to send *initiate messages* again and new transponder task B will be created, situation in the Figure 45 left.

14.3 Evasion Manoeuvres

All cooperative algorithms described in the document are based on the application of evasion manoeuvre to the place of the collision $A_i \otimes A_j$ – identification of the first and last collision time of the first collision between their fps. $t_1^{A_i \otimes A_j}$ is the *first collision time* between fp_i and fp_j if there is $\mathbf{col}(fp_i, fp_j, t_1) = 1$ and $\forall t : t < t_1$ is $\mathbf{col}(fp_i, fp_j, t) = 0$. The *last collision time* $t_2^{A_i \otimes A_j}$ is defined by $\forall t : t_1 \leq t \leq t_2$ is $\mathbf{col}(fp_i, fp_j, t) = 1$ and for $t = \lim_{\delta \to 0_+} (t_2 + \delta)$ is $\mathbf{col}(fp_i, fp_j, t) = 0$. For $\max(rsz_i, rsz_j) > 0$ is always $t_2 > t_1$. Note that the times $t_1^{A_i \otimes A_j}$ and $t_2^{A_i \otimes A_j}$ are the same from both perspectives A_i and A_j, but the $\mathbf{p}(fp_i, t_1) \neq \mathbf{p}(fp_j, t_2)$.

There are defined seven *evasion manoeuvres* $\mathcal{EM}_i = \{\mathbf{em}_L, \mathbf{em}_R, \mathbf{em}_U, \mathbf{em}_D, \mathbf{em}_F, \mathbf{em}_S, \mathbf{em}_0\}$: left, right, climb up, descend down, fly faster, fly slower and leave plan as it is. The set $\mathcal{EM}_i$ can contain different manoeuvres for each airplane $A_i \in \mathcal{A}$, but there must be included $\mathbf{em}_0$ for all planes. The evasion manoeuvre can be seen as a function applied to the flight plan with time specification and returning new changed flight plan. The simplest one is defined as $fp_i = \mathbf{em}_0(fp_i, p, t, t_1, t_2)$. It returns exactly the same fp_i and is used for the simplification of cases where airplane A_i can also do nothing in CA algorithm. All other $\mathbf{em}$s return changed fp'_i respecting constraints given for flight plan and changes elements only from the specified time t – time known from definition of *encounter* en_k. All manoeuvres are applicable only if $t < t_1 < t_2$ and have also specified application strength parametrization p for the evasion manoeuvre.

The application of the right evasion manoeuver to the fp_i is shown in the Figure 46. It changes fp_i from time t that the new fp'_i passes through the points specified by moved $\mathbf{p}(fp_i, t_1)$ and $\mathbf{p}(fp_i, t_2)$ to the right side. The size of the shift is given by the parameter p – for larger p the

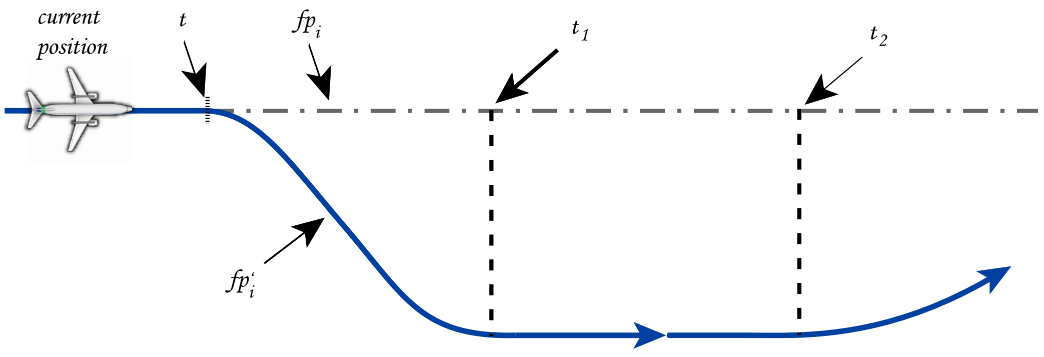


Figure 46: Application of right evasion manoeuvre – $fp'_i = \mathbf{em}_R(fp_i, p, t, t_1, t_2)$

evasion manoeuvre makes larger evasion. The $\mathbf{em}_L, \mathbf{em}_U$ and $\mathbf{em}_D$ are defined similarly to $\mathbf{em}_R$ only with different shift direction of the points to the appropriate side. The velocity changing manoeuvres $\mathbf{em}_F$ and $\mathbf{em}_S$ change the flight plan by time t'_1 that $\mathbf{p}(fp'_i, t'_1) = \mathbf{p}(fp_i, t_1)$. For the fly faster manoeuvre is $t'_1 < t_1$ and for the fly slower $t'_1 > t_1$. The application of $\mathbf{em}_F$ manoeuvre is restricted by the maximal flying velocity of the plane. The $\mathbf{em}_S$ manoeuvre is not restricted, because there can be inserted holding orbit if the minimal flying velocity is reached. The evasion manoeuvres can be combined together by their sequential application.

14.4 Rule-Based Collision Avoidance

It is the domain dependent algorithm inherited from the common collision solver. The collision avoidance mechanism is invoked from the transponder task within the collision avoidance block, Section 14.2. First, the type of the collision between the airplanes is identified. The collision type is determined on the basis of the angle between the direction vectors of the concerned planes at **time 1** (first collision point found by the collision detector, Section 14.1) projected to the ground plane (defined by X and Y axis), see the Figure 47.

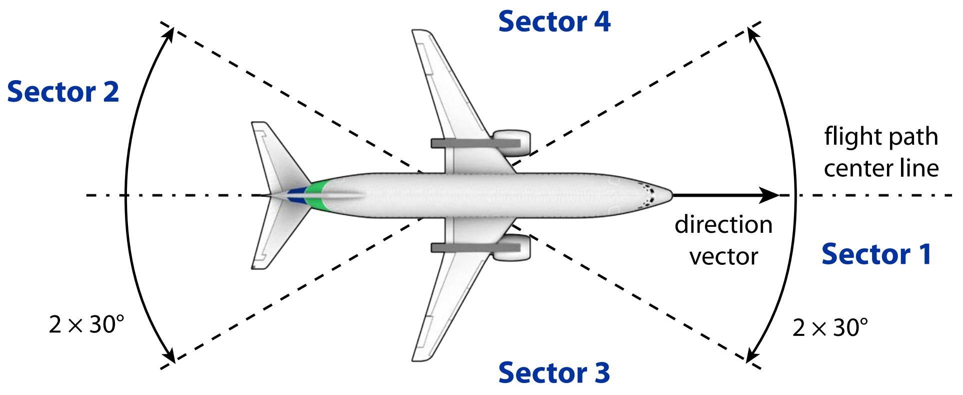


Figure 47: Identification of the collision type in the Rule-based collision avoidance

Depending on the computed angle the plane `B` is relatively to the plane `A` it fits into one of the four sectors. Depending on this sector, one of the following rules is applied on the flight plan of plane `A` to avoid the collision:

- **Sector 1** – head-on collision, in this case the planes avoid each other on the right side of the second one. The plane flight plan is changed as shown in the Figure 46. The pilot agent shifts the points in the **time 1** and **time 2** perpendicularly to the old direction vector to the right. Distance between the previous and the new point is equal to the minimum of the safety ranges. After **time 2** flight plan will continue shortest way to the next mission way-point.
- **Sector 2** – back collision, there are two subcases: i) the aircraft which is in front of the second one is faster – aircrafts do not change current flight plans. ii) The back and faster plane changes its flight plan so it will pass the front plane on the right side without endangering the front one. The flight plan is similar to that in the Figure 46. Again the back plane shifts the old points in the **time 1** and **time 2** perpendicularly to the old direction vector to the right at the distance at least 1.1 times of the safety range.
- **Sector 3** – the side collision when the other plane has traffic priority. The aircraft needs to slow down its speed so that it reaches the first collision point later than the second airplane. If this is not possible due to the minimal flight speed defined for each plane type, the aircraft slows down as much as possible and shifts the flight point from the first collision point to the right so that there is no collision between their flight plans.
- **Sector 4** – side collision when this plane has traffic priority. The aircraft changes its flight plan by accelerating up to its maximal flight speed so that it passes the collision point before the other airplane. The plane only accelerates as much as needed.

The above rule-based changes to the flight plan are done by both planes independently because the second aircraft detects the possible collision with the first plane from its point of view. After applying the changes to the aircraft's flight plan, it sends an updated local flight plan part to all subscribers (planes located in its vicinity). The change is also verified against all other known flight plans of all aircrafts monitored by the board radar system. If there is another collision detected, new changes are applied.

The pilot agent internally uses the *flight plan wrapper* interface for the manipulation with its flight plan. The change requests are handled as a special set of solver time-constrained way-points. Special handling algorithm implements the application of a new change that overrides the old one. The algorithm decides whether an older solver way-point should be removed or not.

The finding of the stable solution for more than two planes with the collision is given by the fact that all airplanes use the rule-based collision avoidance method and also use the same set of collision avoidance rules. In other cases it cannot be guaranteed that the negotiation among more planes will be finite.

14.5 Iterative Peer-to-Peer Collision Avoidance

This section introduces the *iterative peer-to-peer CA* used as a provider of comparison result for multi-party CA. The algorithm solves an encounter $en_k = \langle t, \{fp_i\}_{A_i \in \mathcal{A}_\mathcal{C}} \rangle$ by the selection of the most important collision airplane pairs $\mathcal{I} = \{A_1 \bigotimes A_2, A_3 \bigotimes A_4 \ldots\}$ where each airplane from $\mathcal{A}_\mathcal{C}$ can be included only once in $\mathcal{I}$. Identification of the set $\mathcal{I}$ is done in the distributed manner. Each $A_i \in \mathcal{A}_\mathcal{C}$ selects its opponent A_j from local view of the encounter $en_k^{A_i} = \langle t, \{fp_i\} \bigcup \{\widehat{fp}_j\}_{A_j \in \mathcal{A}_\mathcal{C} \setminus A_i} \rangle$ (see Section 14.1) using

$$\arg\min_{A_j \in \mathcal{A}_\mathcal{C}} \arg\min_{t_1} \mathrm{col}(fp_i, \widehat{fp}_j, t_1) = 1 \ . \tag{1}$$

Each $A_i \in \mathrm{I}$ starts the *pair negotiation on removing collision* (PNRC) with its colliding opponent A_j. If there is a conflict – A_i selects A_j which is already negotiating with other airplane A_k – A_j will finish its current negotiation if and only if $t_1^{A_i \otimes A_j} < t_1^{A_j \otimes A_k}$. Note that for the same times the first selected opponent by A_j will stay. The encounter in which A_j is included can be changed during its PNRC. A_j stops current PNRC if the solved collision no more exists within new encounter or there is identified more important opponent to A_j.

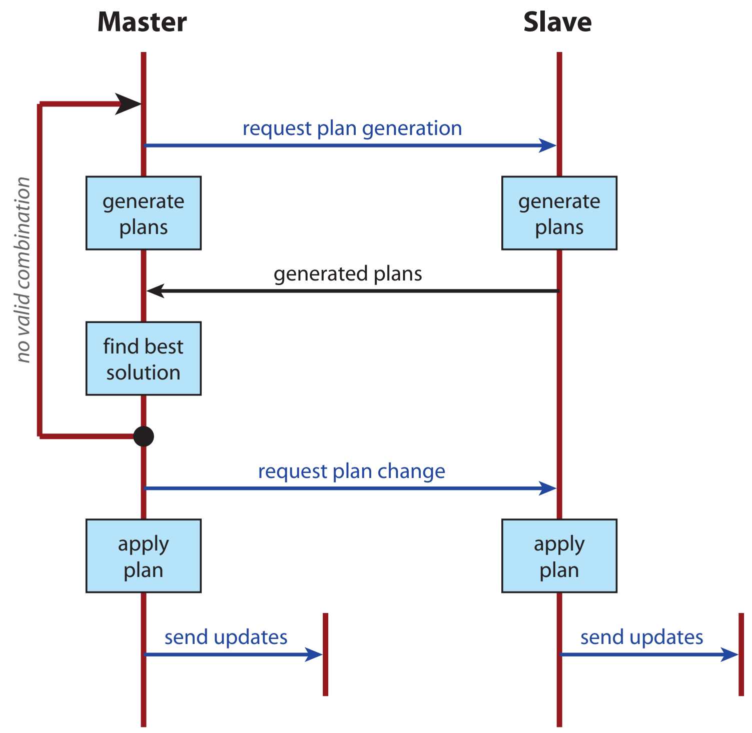


Figure 48: The pair negotiation about removing collision (PNRC) during iterative peer-to-peer collision avoidance

Within pair negotiation about removing collision (PNRC) (see the Figure 48) these airplane pairs $A_i \bigotimes A_j \in \mathcal{I}$ prepare a set of possible changed flight plans with their utility value

$$\mathcal{F}_i = \{\langle fp'_{i1}, \mathbf{u}_i(fp'_{i1})\rangle, \langle fp'_{i2}, \mathbf{u}_i(fp'_{i2})\rangle \dots\} \,. \tag{2}$$

First, the flight plans $\langle fp'_i, \mathbf{u}_i(fp'_i)\rangle \in \mathcal{F}_i$ are given by the application of all $\mathbf{em} \in \mathcal{EM}_i$ to the place of collision $A_i \otimes A_j$ using lowest parametrization p first, see the Section 14.3. Note that the set $\mathcal{EM}_i$ can contain different manoeuvres for each airplane and there is always included *leave plan as it is* manoeuvre $\mathbf{em}_0$ for all airplanes. The changed flight plan fp'_i of $\mathbf{em}$ application is included in $\mathcal{F}_i$ if and only if

$$\forall A_j \in \widetilde{\mathcal{A}_i} \;\text{ is }\; (\arg\min_t \mathrm{col}(fp'_i, \widehat{fp}_j, t) = 1) > t_1^{A_i \otimes A_j} \,. \tag{3}$$

Then both planes A_i and A_j exchange their $\widehat{\mathcal{F}}$ – local future parts of proposed changes with their utility values for next δ interval, see the Section 14.1. The utilities stored in $\widehat{\mathcal{F}}$ are counted for the whole flight plan. A_i prepares the negotiation set

$$\begin{aligned} \mathcal{S}_i^{A_i \otimes A_j} \;=\;& \{\langle fp'_{i1}, \mathbf{u}_i(fp'_{i1}), \widehat{fp}'_{j1}, \mathbf{u}_j(fp'_{j1})\rangle, \\ & \langle fp'_{i1}, \mathbf{u}_i(fp'_{i1}), \widehat{fp}'_{j2}, \mathbf{u}_j(fp'_{j2})\rangle \dots\} \end{aligned} \tag{4}$$

as a cartesian product of $\mathcal{F}_i$ and $\widehat{\mathcal{F}}_j$. Each tuple $\langle fp'_{ik}, \mathbf{u}_i(fp'_{ik}), \widehat{fp}'_{jl}, \mathbf{u}_j(fp'_{jl})\rangle \in \mathcal{S}_i^{A_i \otimes A_j}$ is included if and only if

$$(\arg\min_t \mathrm{col}(fp'_{ik}, \widehat{fp}'_{jl}, t) = 1) > t_1^{A_i \otimes A_j} \ . \tag{5}$$

If the negotiation set $\mathcal{S}^{A_i \otimes A_j}$ is empty, A_i adds to the $\mathcal{F}_i$ flight plans as a result of the application of the evasion manoeuvres $\mathbf{em} \in \mathcal{EM}_i$ using next larger parametrization p. This is done by both A_i and A_j and new $\mathcal{S}_i^{A_i \otimes A_j}$ is generated. The whole process is repeated until the negotiation set holds at least one element.

A_j does the same from its perspective and its view of negotiation set $\mathcal{S}_j^{A_i \otimes A_j}$ holds equivalent elements as $\mathcal{S}_i^{A_i \otimes A_j}$, but has full flight plan for fp'_j and local future parts for fp'_i. Both airplanes A_i resp. A_j propose the solution

$$\begin{array}{ll} \underset{\langle fp'_{ik}, \mathbf{u}_i(fp'_{ik}), \widehat{fp}'_{jl}, \mathbf{u}_j(fp'_{jl})\rangle \in \mathcal{F}_i}{\arg\max} & \mathbf{u}_i(fp'_{ik}) + \mathbf{u}_j(fp'_{jl}) \quad \text{resp.} \\ \underset{\langle fp'_{jl}, \mathbf{u}_j(fp'_{jl}), \widehat{fp}'_{ik}, \mathbf{u}_i(fp'_{ik})\rangle \in \mathcal{F}_j}{\arg\max} & \mathbf{u}_i(fp'_{ik}) + \mathbf{u}_j(fp'_{jl}) \ . \end{array} \tag{6}$$

If there exist more candidates with the same value of selection criterion, the proposed solution is selected randomly from these candidates. To agree with one of the two different randomly proposed solutions they (A_i and A_j) can use protocol based on the commitment scheme known from cryptography [6].

After the distributed application of all solutions for $A_i \otimes A_j \in \mathcal{I}$ the last encounter is partially modified and new one is detected by the *local cooperative detection* (Section 14.1) and described peer-to-peer CA is started again. The restrictions 3 and 5 ensure that the possible application of the solution selected from $\mathcal{S}^{A_i \otimes A_j}$ cannot cause new earlier collision with any airplane's current flight plan of which they are aware than the solved one. This assertion with combination of the creation of $\mathcal{I}$ guarantees the convergency of algorithm [10].

The described iterative peer-to-peer algorithm is also suitable for the application of any other solution selection from the negotiation set than described maximal sum of utilities in criterion 6. For example there can be used classical *monotonic concession protocol* (MCP) [15] – simple protocol for automated agent to agent negotiations in the cooperative domain. The use of such protocol guarantees that both participating agents want to maximize their expected utility. In this case both agents leave only *Pareto optimal* deals in the negotiation set $\mathcal{S}$ and then the agents can use *Zeuthen strategy* [15] for finding the acceptable deal for both agents.

The theoretical study of the algorithm properties as well as its mathematical convergency proof on selected worst case scenario is stated in the final report of the project extension [9]. Besides the convergency there is provided the set of the estimations giving limits of the algorithm iterations for specified number of airplanes. The extension of the report also provides modification of described basic IPPCA algorithm. It is extended with tendencies restrictions reducing the number of iterations and thus reducing the necessary communication flow among airplanes and makes final flight corridors more simple with lower number of segments. On the other hand as it is shown in that report such advantages are compensated by slightly longer final trajectories.

The described IPPCA algorithm doesn't allow any near miss (safety zone violation) during the negotiation. If the density of the planes in the scenario is so high, it is not able to finish PNRC negotiation and defining collision is simply skipped. This leads to occurrences of the uncontrolled near misses. To be able to handle also such situations the algorithm has been extended with the special behavior enabling near misses in these dense cases. The new feature of the IPPCA tries to minimize possibility of the collision with minimization of allowed near misses. In other words, if it is not possible to solve the situation without any near miss, the algorithm gives solution with minimal safety violation of minimum airplanes which is much safer than uncontrolled near misses. For the detailed description of the extension please refer the final report of the project extension [9].

14.6 Multi-Party Collision Avoidance

This section describes the collision avoidance algorithm based on the creation of groups of airplanes which together solve a collision or collisions. The algorithm removes the iterations known from the iterative peer-to-peer algorithm (described in the Section 14.5) appearing in the multi-collision situations (see the Section 4.4). In a more dense airspace, this approach enables better utilization of the airspace. As a motivation for this approach, we can imagine a situation where two airplanes have a collision $A_i \otimes A_j$, but it is difficult for them to avoid the collision as other airplanes are in the airspace near to them. The situation can be so difficult that they can have only two options, dramatically deviate from their courses, or deviate only slightly but make their flight plans colliding with another airplanes' flight plans. However, they can create a group with the other airplanes and solve the collision $A_i \otimes A_j$ together with them. Basically, we can say that the two colliding airplanes will ask other airplanes to make space for their evasion maneuvers.

The basic idea behind the presented multiparty algorithm is to search the state space of possible applications of sequences of evasion maneuvers on the flight plans of airplanes. The goal of the search is to solve a multi-collision with respect to given criterion evaluating the fitness of the solution. In our experiments we use the sum of flight plan utilities for decimalization of the social welfare. There is no restriction how much evasion maneuvers an airplane can apply to its flight plan. This means that the state space is infinite. The multiparty collision avoidance is motivated by the A* algorithm [12]. The A* algorithm finds the optimal solution in a finite time if there is a solution. Our situation is more difficult, each airplane has its local information about other airplanes. This information can change during the search. This is the reason why we can not use pure the A* algorithm, we can not specify searching space in the beginning of the search. When the new plane appears in the communication range, its flight plan can collide with some airplane in multiparty group. Then the A* algorithm should be restarted because of state space change. Our algorithm just updates states in the open list and continues with the search. This approach removes the loss of the progress of the search by restart. There are no cycles in the state space so the list for storing of already expanded states can be omitted from the algorithm. We will use only open list $\mathcal{O}$ for storing of states generated by the expansion of other states and not yet expanded.

For a given encounter $\langle t, \{fp_i\}_{A_i \in \mathcal{A}_C} \rangle$ a *multiparty group* $\mathcal{G} \supseteq \mathcal{A}_C$ is a set of airplanes whose flight plans are involved in a solution search process. The goal of the group is to find a solution for the encounter. Note that solution provided by the multi-party algorithm has to contain flight plans for airplanes A_C, but usually it will contain additional flight plans for airplanes located nearby. A *state* s is a set $\{s_i\}_{A_i \in \mathcal{G}}$, where $s_i = \langle \widehat{fp}_i, \mathbf{u}_i(fp_i) \rangle$ is a tuple containing the local flight plan $\widehat{fp}^s$ of airplane A_i for state s and its utility value computed by airplane A_i from the full flight plan fp_i^s. $\mathcal{C}^s$ is a set of all collisions among flight plans from s and $\mathbf{g}^s$ is the cost of application of evasion maneuvers. An *initial state* s_0 contains current local flight plans $\{\widehat{fp}_i\}_{A_i \in \mathcal{A}_C}$(plans before applying any evasion maneuver), $C^{s_0} = C$ where C is initial multi-collision for the starting encounter and $\mathbf{g}^{s_0} = 0$. $\mathbf{g}^s$ is recursively defined. For the given state s and its child state $s' = \{\widehat{fp}_i^{s'}\}_{A_i \in I} \cup \{\widehat{fp}_i^{s}\}_{A_i \in \mathcal{G} \setminus I}$ is the set of changed local flight plans by application of evasion united with the set of their predecessors, then

$$\mathbf{g}(s') = \mathbf{g}(s) + \sum_{A_i \in I} \mathbf{u}_i(fp_i^s) - \sum_{A_i \in I} \mathbf{u}_i(fp_i^{s'}). \tag{7}$$

The set $\mathcal{I} \subseteq \mathcal{G}$ holds exactly two airplanes which apply the evasions to remove selected collision as described later. In other words, $\mathbf{g}(s') - \mathbf{g}(s)$ is the cost of the application of evasion maneuvers with goal to remove one single collision of two flight plans. The *final state* s_f is a state which is the solution of an encounter, basically a set of non-colliding flight plans. For a given state s, the *evaluation function* $\mathbf{f}(s) = \mathbf{g}(s) + \mathbf{h}(s)$ consists of the cost of the application of evasion maneuvers $\mathbf{g}(s)$ and heuristic function $\mathbf{h}(s)$ which estimates how many more changes are needed to remove all collisions from the flight plans in s.

At the beginning the multiparty group contains only the airplanes which create it: $\mathcal{G} = \mathcal{A}_C$. The searching algorithm of the group proceeds in a cycle until it finds the solution. The state with the lowest value of the evaluation function is selected $s^* = \arg\min_{s \in \mathcal{O}} f(s)$. All flight plans from s^* are checked for collision with local flight plans of airplanes from the set $\mathcal{A} \setminus \mathcal{G}$. Any colliding airplane not yet included in the set $\mathcal{G}$ is asked to join the group. If the airplane A_y joins the group, then its actual local flight plan is added to all states in $\mathcal{O}$ and to state s^*. Precisely, any state $s \in \mathcal{O}$ is replaced by a new state $s \cup \langle \widehat{fp}_y, \mathbf{u}_y(fp_y)\rangle$ (similarly for s^*). Note that the cost of the state $\mathbf{g}(s)$ does not change by addition of the new flight plan, there is no application of evasion maneuvers. If the state s is the final state, the algorithm finishes and the planes in the multiparty group change their actual flight plans to the flight plans $\{fp_i^{s_f}\}_{A_i \in \mathcal{G}}$ which correspond to local flight plans in the chosen final state s_f. From the description of the algorithm below, it is clear that each airplane A_x knows for each generated local flight plan $\widehat{fp}_i^{s_f}$ its full version $fp_i^{s_f}$.

In the other case - when s^* is not the final state, the pair of airplanes A_i, A_j with the earliest collision in the state s^* is selected by

$$\arg\min_{A_i, A_j \in \mathcal{G}, A_i \neq A_j} t_1^{fp_i^{s^*} \otimes fp_j^{s^*}} \tag{8}$$

and these airplanes A_i, A_j generate combinations $S^{fp_i^{s^*} \otimes fp_j^{s^*}}$ of flight plans to remove their (earliest) collision. The airplanes prepare sets F_i resp. F_j of possible changed flight plans with their utility value

$$\mathcal{F}_i = \{\langle fp'_{i1}, \mathbf{u}_i(fp'_{i1})\rangle, \langle fp'_{i2}, \mathbf{u}_i(fp'_{i2})\rangle \ldots\} \tag{9}$$

and their local versions $\widehat{F}_i$ resp. $\widehat{F}_j$ for next δ interval, see the Section 14.1,

$$\widehat{\mathcal{F}}_i = \{\langle \widehat{fp}'_{i1}, \mathbf{u}_i(fp'_{i1})\rangle, \langle \widehat{fp}'_{i2}, \mathbf{u}_i(fp'_{i2})\rangle \ldots\} \ . \tag{10}$$

The flight plans $\langle fp'_i, \mathbf{u}_i(fp'_i)\rangle \in \mathcal{F}_i$ are given by the application of all $\mathbf{em} \in \mathcal{EM}_i$ to the place of collision $fp_i^{s^*} \otimes fp_j^{s^*}$ using the lowest parametrization p, see the Section 14.3. Note, that the set $\mathcal{EM}_i$ can contain different manoeuvres for each airplane and there is always included the *leave plan as it is* manoeuvre $\mathbf{em}_0$ for all airplanes. $\mathcal{S}^{fp_i^{s^*} \otimes fp_j^{s^*}}$ is then a subset of combinations of flight plans from $\widehat{\mathcal{F}}_i$ and $\widehat{\mathcal{F}}_j$ which have no collision or the first collision point of the collision is not earlier that the old collision point, precisely

$$\begin{aligned} \mathcal{S}^{fp_i^{s^*} \otimes fp_j^{s^*}} = \quad & \{\{\langle \widehat{fp}'_{ik}, \mathbf{u}_i(fp'_{ik})\rangle \in \widehat{\mathcal{F}}_i, \\ & \langle \widehat{fp}'_{jl}, \mathbf{u}_j(fp'_{jl})\rangle \in \widehat{\mathcal{F}}_j\} : \\ & t_1^{\widehat{fp}'_{ik} \otimes \widehat{fp}'_{jl}} > t_1^{\widehat{fp}_i^{s^*} \otimes \widehat{fp}_j^{s^*}} \ \} \end{aligned} \tag{11}$$

The set of new states $\mathcal{N}$ is created using

$$\begin{aligned}\mathcal{N} = \quad & \{s' = old \cup new| \\ & old = s^* \setminus \{s_i^*, s_j^*\}, new \in \mathcal{S}^{fp_i^{s^*} \otimes fp_j^{s^*}}, \\ & \forall A_x \in \{A_i, A_j\}\ \forall A_y \in \mathcal{G} \setminus \{A_i, A_j\} : \\ & t_1^{\widehat{fp}_x^{new} \otimes \widehat{fp}_y^{old}} > t_1^{\widehat{fp}_i^{s^*} \otimes \widehat{fp}_j^{s^*}} \ \} \ .\end{aligned} \tag{12}$$

New states $\mathcal{N}$ are added to $\mathcal{O}$ and the searching continues with expansion of the state with the smallest value according to the evaluation function.

By default the algorithm uses the zero heuristics. With general utility function for the flight plan, it is possible to have evasion maneuvers that do not change the utility of flight plane and can be used for collision avoidance, so only zero heuristic is admissible [12]. For example assume we have a scenario where two planes have a collision on their perpendicular flight plans and the utility value depends only on the flight plan length. The best solution is when one plane speeds up and another slows down and in this case the utility value is the same as in the initial state.

The algorithm with zero heuristics finds the final state with lowest value of evaluation function which corresponds to the lowest utility loss for the sum of utilities of flight plans. However, in the worst case the expanded state space can grow exponentially with the number of collisions in a multiparty group. We can also use different not admissible heuristic. Such as "collision" heuristic which would combine the main utility criterion with giving more preferences to the states with less collisions. The search process with such heuristics is faster, but its result can be far from the best possible utility.

14.6.1 Interaction of Multi-party groups

When the airplane is asked to join the multiparty group $\mathcal{G}_1$, it can be already participating in another multiparty group $\mathcal{G}_2$. In this case, the airplane checks if $\mathcal{G}_1$ is more important than $\mathcal{G}_2$ and if so, the airplane terminates its interaction with group $\mathcal{G}_2$ and joins the group $\mathcal{G}_1$. When an airplane terminates the interaction with the group, the group is dissolved. The relation of importance of groups is defined according to the earliest collision in their encounter. When the group $\mathcal{G}$ is searching for solution of the encounter $\langle t, \{fp_i\}_{A_i \in \mathcal{A}_C} \rangle$ then the time t_G of the earliest collision is:

$$t_{\mathcal{G}} = \min_{A_i, A_j \in \mathcal{G}, A_i \neq A_j} t_1^{A_i \otimes A_j} \tag{13}$$

The group $\mathcal{G}_1$ *is more important* than $\mathcal{G}_2$ – $\mathcal{G}_1 \succ \mathcal{G}_2$ if and only if $t_{\mathcal{G}_1} < t_{\mathcal{G}_2}$. To make $\succ$ relation total, it must be defined also for situations when $t_{\mathcal{G}_1} = t_{\mathcal{G}_2}$, also it is not important if $t_{\mathcal{G}_1} < t_{\mathcal{G}_2}$ or $t_{\mathcal{G}_1} > t_{\mathcal{G}_2}$. It can be chosen for example randomly, or deterministically - with help of the lexicographic ordering of the string representation of the groups.

15 Non-cooperative Collision Avoidance

In the case when the communication between planes is not possible, it is required to solve the collisions non-cooperatively. This can happen in the situation when e.g. the on-board communication device of the plane is broken, the used collision avoidance systems are not compatible or the other plane intentionally refuses to communicate (an enemy).

There are implemented two methods for handling such situations in the AGENTFLY. The first is based on the non-cooperative object future position prediction with encapsulation to the dynamic no-flight zone used for testing and flight plan replanning, described in the section 15.1. The second non-cooperative method is based on the proportional navigation which provides optimal collision avoidance between two UAAs, described in the Section 15.2.

15.1 NFZ-based Collision Avoidance

The dynamic no-flight zone-based non-cooperative collision avoidance has been designed and implemented in the AGENTFLY system as one of the methods for collision solving. In the contrast with other methods, the described solution is based on the path planning algorithm in order to avoid dynamic no-flight zones, described in the section 6.1. These zones surrounding collisions with non-cooperative objects are regularly updated. The non-cooperative objects are detected using an airplane on-board radar sensing feature.

The algorithm is responsible for the coordination of all operations needed for avoiding potential future collision of the airplane and an object in space the airplane cannot establish communication with. The algorithm is implemented in the form of a special unit (a solver of the multi-layer collision avoidance framework, described in the section 13) that can take part in the process of collision solving within the pilot agent. The scheme of the NFZ-based non-cooperative solver is shown in the Figure 49.

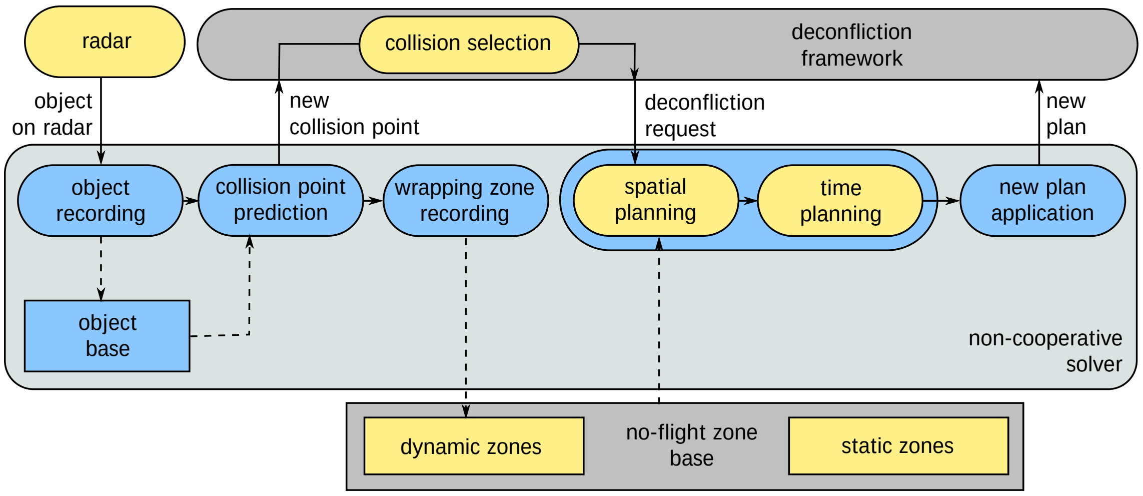


Figure 49: NFZ-based non-cooperative avoidance solver scheme

The event that triggers the collision avoidance loop is the information obtained from the radar describing the position of an unknown object in the area. This object is recorded in the base of non-cooperative objects, unless it's already present there. If the object is already known, its position is updated.

The next step is the prediction of the collision point (Section 15.1.1), an intersection of the flight plan and the non-cooperative object. If no such virtual collision point is found, the loop ends. In the opposite case, the collision point is wrapped by a dynamic no-flight zone and it is passed on to the collision avoidance framework that will decide whether it is necessary to solve the given collision. The solution process may not be executed in case that the collision has already been detected and processed by another solver earlier.

If the collision avoidance framework decides that the collision should be solved, the last step of the non-cooperative solver is executed, i.e. a new flight plan from the current position to the destination point is generated using spatial and time planning. Spatial planning takes into account all static no-flight zones as well as dynamic no-flight zones of all non-cooperating objects (Section 15.1.2). The new plan is then passed to the collision avoidance framework to be handled by the pilot agent.

The described collision avoidance loop is executed for all objects found in the radar scan. This is done periodically for each radar scan (cycle period t_r).

15.1.1 Prediction of the Collision Point

In order to detect a potential future collision it is necessary to find a collision point $\overline{c}$ in the space and time. It is a point defined by an intersection of the current flight plan and the flight trajectory of the non-cooperative object. Since the future trajectory of the non-cooperative object in not known exactly, it must be extrapolated from the object's motion history. There are implemented two future trajectory predictors: *liner* and *predictor based on Taylor series.*

The linear predictor estimates the future trajectory from the following two points: the current position $\overline{s}$ and the position read from the previous radar scan $\overline{s_p}$. The current direction vector $\overline{d}$ of the non-cooperative object is calculated simply as

$$\overline{d} \quad = \quad \overline{s} - \overline{s_p}. \tag{14}$$

The current estimated speed v of the object is calculated as

$$v \quad = \quad \frac{\|\overline{d}\|}{t_r}, \tag{15}$$

where t_r is the radar cycle period.

The linear prediction is not well suitable for predicting future trajectory of the object which quite often changes its direction (e.g. fly along turn element) because the prediction error is so high. Besides linear prediction the AGENTFLY has integrated predictor based on Taylor series. Full three dimensional predictor is realized as three independent Taylor series – one per each coordinate. The following description shows how the prediction works for one coordinate. The position of the monitored object can be viewed as a function of time. The function value in particular time t can be counted from the Taylor series:

$$\tilde{\mathrm{f}}(t) = \sum_{n=0}^{N} \frac{\mathrm{f}^{(n)}(t_0)}{n!}(t - t_0)^n \tag{16}$$

where $\mathrm{f}^{(n)}(t_0)$ denotes the n-th derivative of f at the time t_0 and the zeroth derivative of f is defined to be f itself. The N defines the number of the components in the Taylor series. If the $N = \infty$ then $\mathrm{f}(t) = \tilde{\mathrm{f}}(t)$. For $N < \infty$ the error of prediction is 0 for the time t_0 and rise with the higher difference $t - t_0$.

The AGENTFLY implementation of Taylor series works over the sequence of discrete position samples with time stamps from the radar sensing. The derivative in the equation 16 is replaced by the differences counted from that chain of last position samples. To count n-th difference $n+1$ last points are used. The computation of n-th differences are done in iterative way. The values for the k-th iteration are:

$$\begin{aligned}
\Delta^{(0)}\mathrm{f}(t_k) &= \mathrm{f}(t_k), \\
\Delta^{(1)}\mathrm{f}(t_k) &= \tfrac{\mathrm{f}(t_k)-\mathrm{f}(t_{k-1})}{t_k-t_{k-1}}, \\
\Delta^{(2)}\mathrm{f}(t_k) &= \tfrac{\Delta^{(1)}\mathrm{f}(t_k)-\Delta^{(1)}\mathrm{f}(t_{k-1})}{t_k-t_{k-1}}, \\
&\vdots \\
\Delta^{(n)}\mathrm{f}(t_k) &= \tfrac{\Delta^{(n-1)}\mathrm{f}(t_k)-\Delta^{(n-1)}\mathrm{f}(t_{k-1})}{t_k-t_{k-1}}
\end{aligned} \tag{17}$$

where $\Delta^{(n)}\mathrm{f}(t_k)$ denotes the n-th differences of f at the time t_k. If the order of the predictor (and the number of components in the Taylor series) is $N = 1$, then this predictor is the same as the linear one.

The prediction of the future position from the limited number of the last observed positions of the object based on the separate prediction of each coordinate component with finite prediction order (finite number of last known points) provides prediction with object movement higher that the maximum flight velocity. The function providing the predicted position contains this velocity restrictions and if the plain Taylor series value is out of the bound, the function uses only the shape of predicted trajectory and discards the velocity of the movement by its bounding.

In order to find the collision point, the flight plan and predicted trajectory are iterated in the future time. In each iteration the distance between the predicted position and the position of the airplane given by the current flight plan. If this distance is shorter than defined safety zone, the collision point is found.

The size of the no-flight zone derived from the possible future flight trajectory (see the Section 15.1.2) of the non-cooperative object is stretched so that it is twice as long as the time from the current position to the collision point.

15.1.2 Dynamic No-flight Zone Shapes

The linear predictor can be coupled either with the ellipsoid zone shape or with the shape given by the future object position for given time interval, see the Figure 50. The shape of the dynamic no-flight zones for non-cooperative object is derived from the possible future flight trajectory. The trajectory takes into account the minimal turning radius r_{min}, maximal climbing and descending angle φ_{max} and the prediction time t_p.

The second zone is internally represented as with the binary octant tree (described in the Section 4.3). The advantage of octant trees is the efficient data storage and fast point and line tests. Disadvantages include slow construction of octant trees and rough sampling of cells. For the purposes of NFZ-based non-cooperative collision avoidance it is necessary to solve the problem of fast translation, rotation and deformation of relatively small octant subtrees used as dynamic no-flight zones of non-cooperative objects.

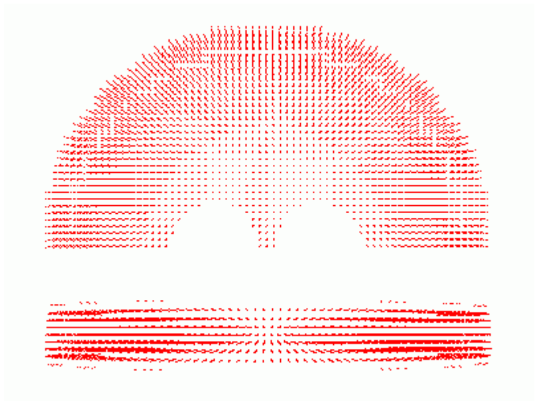


Figure 50: Shape of the dynamic no-flight zone

Since complete rebuilding of an octant tree for each specific identified collision is too slow, a different solution was designed. It is based on separation of all particular octant trees (for all collision no-flight zones). The shape of the zone depends on the several configuration parameters which are coupled with the type of the object. Thus, in the AGENTFLY there are defined such zone for each known object type and the subsequent transformation is applied to the basic octant tree. For such purposes the transformation node in the airspace representation is used (Section 4.3).

The predictor based on Taylor series is restricted to use only ellipsoid dynamic no-flight zone. This restriction is given by the non-linear shape of the predicted trajectory from the current position of the object to the collision position. Theoretically it is possible to construct such zone also for non-linear trajectory, but it will be suitable only for that trajectory shape and cannot be reused for other collisions. Due to the slow construction of that zone this solution cannot be applied. The ellipsoid dynamic zone is constructed using basic geometrical object representation (see the Section 4.3) and its center is inserted just to the place of the detected collision. The size of the ellipsoid axis are then inferred from the velocities (both the airplane and object) in the collision point and distance of the collision point from the current airplane position. The error of the prediction rises along for later values. Thus the higher size of the ellipsoid partially compensate this effect. For higher speed and further collision the ellipsoid is larger and vice-versa.

15.2 Optimal Proportional Navigation Algorithm

For purposes of experiments of non-cooperative collision avoidance (Section 17.2) it was necessary to extend the AGENTFLY system with an implementation of a reference collision avoidance algorithm. The Proportional Navigation-Based Optimal Collision Avoidance for UAAs [4] was chosen, or in short Optimal Proportional Navigation.

The optimal proportional navigation algorithm tries to keep a predefined safe distance between an airplane and an obstacle (non-cooperative one tracked on its on-board radar). This is achieved by directing the airplane towards the edge of the safety zone and an appropriate acceleration. If the airplane gets into such configuration with the obstacle in which no future collision exists, the airplane control algorithm switches its mode from collision avoidance to navigation towards the destination point. The algorithm pseudo-code is as follows:

```
do while (airplane has not reached its destination)
   calculate relative speed $v_{rel} = v - v_T$
   if (relative speed vector is outside the obstacle cone)
      switch mode to navigation towards destination
   else
      switch mode to deconfliction
   end
end
```

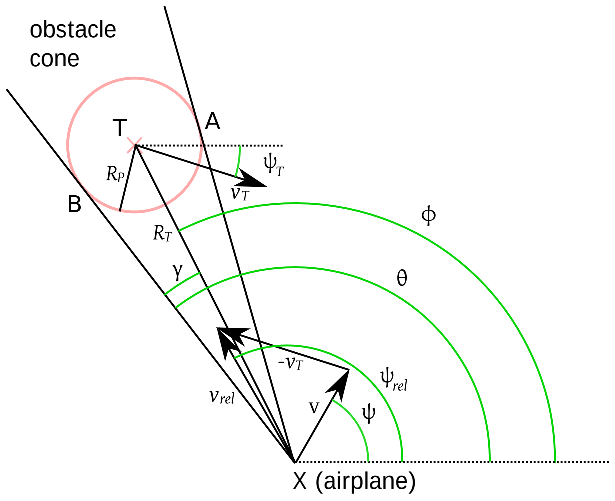


Figure 51: Scheme of the situation handled by the optimal proportional navigation

For purposes of collision avoidance, a vector $\overrightarrow{\mathbf{XB}}$ is defined. The vector points to the edge of the safety zone (see Fig. 51). The relative speed vector is dragged towards the direction of this vector in order to reach the airplane direction required for successful obstacle avoidance. Proportional navigation control can be expressed as follows:

$$a = N v_{rel} \dot{\theta}, \tag{18}$$

where a is acceleration, v_{rel} is relative speed, θ is the angle of vector $\overrightarrow{\mathbf{XB}}$ and N is the proportional navigation constant. The fore-mentioned formula corresponds to the classical proportional navigation. Additional angle calculation follows:

$$\dot{\theta} = \dot{\phi} + \dot{\gamma}, \tag{19}$$

$$\dot{\theta} = -\left(\frac{v_{rel} \sin \psi_{rel}}{R_T \cos \phi} + \frac{\dot{R_T}}{R_T} (tan\phi + tan\gamma) \right). \tag{20}$$

The calculation therefore depends on the fore-mentioned angles (see Fig. 51).

We define three sufficient conditions for switching to the navigation-to-destination mode:

1. distance between the airplane and the obstacle is greater than the safety zone ($R_T \geq R_P$)
2. relative speed vector is outside the obstacle cone
3. the obstacle is located beyond the relative speed vector

The last step is the calculation of the optimal proportional navigation constant N. It is based on calculus of variations and the optimization theory. The result is a system of equations. By solving this system we obtain the constant N:

$$\mu = \frac{v_T \cos(\theta_f - \psi_T)}{v_{rel}}, \tag{21}$$

$$N = 1 - \mu \pm \sqrt{\frac{K}{1-K} + (1-\mu)^2}, \tag{22}$$

$$\theta_f = \frac{N\theta_0 - \psi_{rel}}{N-1}, \tag{23}$$

where K is the integration constant and θ_0 and θ_f are the limit values. Moreover, the constant N must fulfill the condition $N > 1$, which is based on analysis of convergence of the algorithm.

All proofs and additional information about the algorithm can be found in [4].

16 Collective Flight

The *collective flight* functions in the AGENTFLY system provides capability for the automatic synchronization of the group of independent airplanes. Each airplane flies along its given mission until there is defined special way-point in its mission. The name of this way-point uniquely specifies that the airplane should continue its flight through defined corridor. If the next way-point is such one, the airplane enables the control modules providing the collective flight functions. The airplane has got the configuration of airplanes that are in the same group. Using the knowledge about the group airplane starts the negotiation with others and identifies how many of them have the intention to use the same corridor. Then all such airplanes need to make the flight formation to fly via the defined corridor. At the end of the corridor the airplanes break up the formation package and each follows the next part of its mission specification. During the whole airplane mission there can be specified several corridors to pass. The current AGENTFLY implementation allows only straight corridors paths – the airplanes flying in the formation cannot make any turns until the decomposition of the formation package.

The AGENTFLY system supports also more complicated situations where there are more defined groups operating in the same area and have intention to fly through the same corridors – corridors are uniquely given by their name id. For these situations each corridor has defined the separation time interval and a set of alternative rendezvous points. The next group package can enter the corridor after defined separation time. If it is not possible to fly via corridor when all airplanes are ready they take holding orbit at one of the rendezvous point.

The described functionality is implemented in two connected modules. The first module does the group synchronization (Section 16.1) and the second one provides the precise composition, flight and decomposition of the formation package (Section 16.2).

16.1 Group Synchronization

The group module of the collective flight is responsible for the initial group synchronization before the formation package setup phase. It provides inner group synchronization during which it identifies airplanes having the same intention to fly through the same corridor. First the airplanes go to the selected rendezvous location. If at least one of the airplanes is delayed due to any reason, others must wait for it applying holding orbit at the rendezvous point. During the holding phase they still monitor if the missing airplanes will come. The missing airplane can reject the group participation. E.g. it needs to perform the refuelling before the next part of the missing but the tanker doesn't come to the tank airspace. Thus, in such situation the airplane identifies that it has fuel just for return back to the base and its rest part of the mission cannot be accomplished. If all airplanes in the group are ready and others reject the group participation, the group module invokes the formation module and starts the next phase of the forming formation package, see the Section 16.2.

Besides inner group synchronization the module provides also outer group-to-group negotiations. There is the predefined set of alternative rendezvous location for each corridor. If the group of airplanes identifies that another group is occupying the same plane due to the use of the same corridor, the group starts the negotiation with the other group and depending on their mutual priorities (now given by the pre-planned corridor use time window) one of the group decides to take other rendezvous location. The group with higher priority takes the location nearer to the corridor entry point. Also if the group is already waiting in the holding orbits, the group is able to move to the alternative location and perform waiting there. The group with the lower priority checks the time when the previous one enters the corridor and its own entrance is then planned at least defined corridor separation duration later.

The implementation of the group module in the AGENTFLY utilizes the social dominance concept and communication routing as it was developed in one of the previous AFRL project – FA8655-02-M-4057 Inaccessibility in Multi-agent Systems [8]. There is not apriori selected leader responsible for the group management and the leader is designated depending on the unique priority of the airplane in the group using the dominance concept. This allows simply to handle the situation when the current leader of the group has problems disqualifying it from the leader position – e.g. it cannot communicate with others due to the transceiver error or it is damaged or lost. In such case the other member of the group automatically becomes its leader. The communication among airplanes has limited range and the synchronization communication in the group is not always possible in one hop. This requires that the group is able to somehow reroute commands among members from one location to the other.

The combination of two described concepts allows to handle very complicated cases. For example the whole group cannot communicate although they are using routing. In that case the group is split into several independent subsets which cannot communicate together. Then each subset has own set leader which provides synchronization the group subset. The social dominance concept automatically handles the situation when two subsets are merged – there exists the communication path among their members. As well as the situation when one subset is split to more subsets. The specification of the rendezvous location guarantees that all subsets become one group at that position and then all airplanes act as one coherent group package.

16.1.1 Synchronization Messages Types

In the group module there are exist two groups of the synchronization messages:

- *The group leader → the members* – messages from the current group leader must reach all group members. Such a message is realized as a multi-recipient message in the $\mathcal{A}$-**globe** platform. The leader selects as a recipients all members which are reachable directly by it. The set of the reachable neighbors is provided by the directory services of the system. The leader sends the message to them. The message itself holds two data structures used for the routing: the set of already addressed members and the path back to the group leader. If any of the members is aware of the existence and accessibility of not addressed members it simply forwards the message to them. Before the rerouting the message it updates the message routing structures. Using this mechanism the message reaches all members from the mutually accessible group of the airplanes. And as the side effect of the multi-recipient routing each of the members knows the actual communication path back to the group leader.

  The communication messages of each type are also remembered by the member and if the new member becomes accessible they are immediately forwarded to it.

- *The member → the group leader* – this kind of messages is simply sent along the known path to the group leader. If the airplane to which the message should be send is not accessible the airplane switches the message to the broadcast mode (similar to the routing in the first group) to reach the group leader if it is present.

16.1.2 Group Coordination Algorithm

Each member of the group knows if it acts as a group leader now. If it is not a group leader, it monitors the communication from the leader to check the leader presence and the dominance of the leader. If there is no message from leader for defined timeout period or the leader priority is lower, the airplane automatically switches itself to the leader mode. On the other hand, if the airplane is in the leader mode and it receives message from the other leader with higher priority, it automatically switches its mode to the participant mode and accepts all commands from the leader.

There are defined following control messages for the group synchronization:

- *Keep alive* – this message is periodically sent by the group leader to ensure that all members are still aware about its existence. The second function of this message is that the members update their communication path to the leader. The time interval for this message is coupled with the monitoring timeout defined for the participant mode.
- *Use rendezvous* – once the airplane becomes the leader of the group it selects one of the alternative rendezvous locations for the corridor and sends the selection to the participants. Every time when the leader changes the selection it dispatches new choice using this message. The period for the *keep alive* message is initialized again after this message because this one provides the same function to the members. All airplanes receiving this command simply change their target to the new rendezvous location and if they are already waiting in the holding orbit in the other location they move to the new one and apply the holding orbit there.
- *Wait for you* – this message is sent once by the group leader in the case that it has already reached the rendezvous location. Thus every time when an airplane becomes the group leader, it sends *use rendezvous* command and *wait for you* command if it is already ready for the formation composition. The participant receiving this command starts periodical dispatch of the group status information (*ready too* or *wait for me*) to the leader. The group leader keeps the info about all group members states. If all members are ready, the leader invokes the formation module to start formation composition phase (see the Section 16.2). Otherwise the members are still waiting for others and wait in the holding orbits in the rendezvous position.
- *Wait for me* – this message is sent from the member to the leader after receiving *wait for you* command. This message is periodically sent to the leader until the airplane reaches the rendezvous location. The leader thus monitors the existence and intention to still participate in the package by the member. If the leader doesn't receive *wait for me* or I am ready message from the member in defined timeout interval it marks the airplane as not available and doesn't wait for it.
- *I am ready* – this message is sent by the group participant as opposite to the *wait for me*. It provides information that the participant is ready for the formation composition phase.
- *Formation done* – the group leader informs all members that the formation module finishes its formation setup negotiations. The message finalizes the group synchronization and no other messages are sent in this phase. The *formation done* contains the final corridor entrance time.

All described messages are sent mainly for the inner group synchronization among group airplanes. But some of them are also used for outer group-to-group synchronization. The *use rendezvous* is also sent to the identified airplane which doesn't belong to my group. The airplane that received such message and participates in the other group (with different group name id) reroutes this message to its group leader. The leader checks if the the priority of the own group is higher or lower and then decides if own rendezvous location selection is updated. If not due to the higher priority, the second group leader receives its selection too and thus the rendezvous change is its responsibility.

Besides this rendezvous group-to-group selection the *formation done* message is exchanged among the groups too. The group from the nearest rendezvous location enters the corridor first. And the other groups wait until they receive the *formation done* message from that prioritized group with corridor entrance time. The group with lower priority still waits in the holding orbit and then the group leader tries to setup formation package for the received time shifted by the corridor separation time interval.

16.2 Formation Composition And Decomposition

The formation module is responsible for the negotiation during the formation setup phase. It needs to find the exact contact time for all autonomous airplanes and also to select the position in the formation pattern respecting the defined constraints. In the formation flight all airplanes are very close each other and thus the criterion for their separation used in the collision avoidance is not satisfied. The avoidance module is disabled in this case and the responsibility for the collision-free flight paths is on the formation module too. Besides this initial setup phase the module does the flight in the formation package execution.

The formation flight, as it has been implemented so far, consists of four steps. The first step is the flight from the current position to the location where the formation assemblage starts. The second step is the formation assemblage, third step is the formation flight and finally the fourth step is the formation split-up.

16.2.1 Formation slots assignment

When the UAAs which pass through the rendezvous point are ready for starting negotiation about formation composition. If all members are there the current leader gives the request for the formation assemblage to its *Formation Manager* module.This request consists of the formation corridor start position, formation corridor end position, addresses of the formation members and start time of the formation (time, when the formation is to be assembled in the start position). Initially the time is given by the pre-planned package time. This time is later adjusted depending on the current position of all members their constraints (airspace, flight model, etc.).

Formation members are divided into three categories according the type of the airplane in order to set the specific formation slots to the formation members. These categories are escort coverage, heavy and unspecified category. The members of the escort coverage category are placed on the edge of the formation. The members of the heavy category are placed in the very back part of the formation because of the turbulence they produce. The remaining slots are assigned to the other types of airplanes. The restriction which type of airplane belongs to which category is specified in the current manager configuration.

The algorithm first counts the number of the airplanes in each category. Then it needs to determine which slots are going to be assigned to escort coverage category. It starts with the position P0 (see the Figure 52) and then it assigns the positions on the edges - P1, P2, P3, P5 etc.

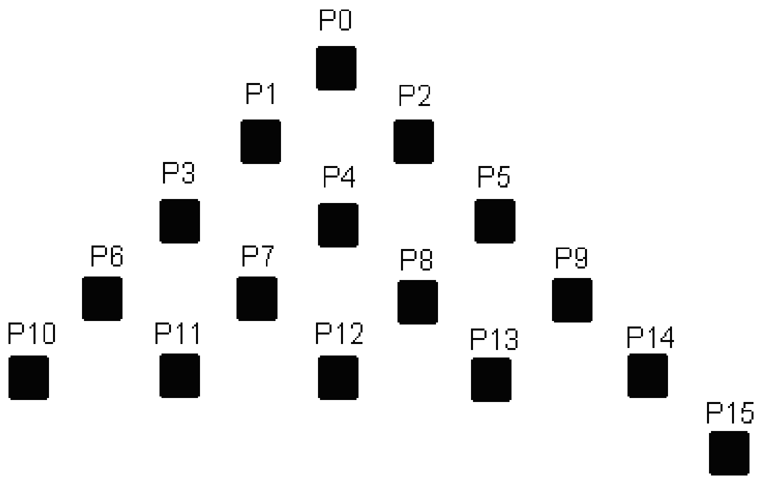


Figure 52: Formation pattern

until the number of positions assigned to the escort coverage category is equal to the number of airplanes in this category.

In the next step, the algorithm assigns the very last positions e.g. P11, P13 to the heavy group. Which slots in the formation will be assigned depends on the total number of the airplanes in the formation and on the size of escort coverage group and heavy group. In the very end, the slots for the unspecified group are assigned. The rule is that if there are fewer members in the formation than 16, the lower positions are always assigned.

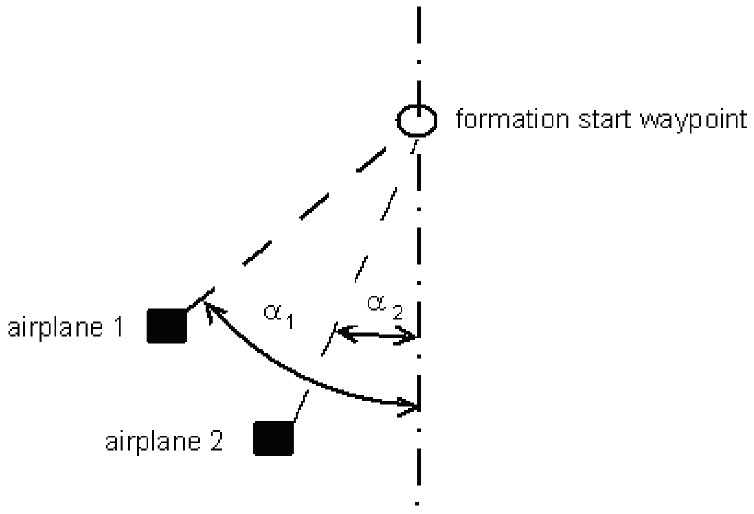


Figure 53: Identification of the deviation angle for the initial assignment of the positions to all airplanes.

After the assignment of the slots to the groups, slots are assigned to specific airplanes in every group. During the slot assignment, the position of the airplane in the moment of the slot assignment is crucial. The formation coordinator receives actual position from every airplane and then it computes the angle between the vector $\overline{v}$ and the direction of the formation corridor. Vector $\overline{v}$ is determined by the actual position of the airplane and the formation corridor start position, see the Figure 53.

The angle value is used for the slot assignment to the group members in order to minimize the number of intersections of the flight plans when the airplanes are flying from their current position to the formation corridor start position. The algorithm sorts the members of the group according the angle value in ascending order. Then it starts assigning the most left formation slots to the sorted airplanes until all positions for respective group are assigned. This is repeated for all three groups, until all slots are assigned.

16.2.2 Flight Path Planning

When the positions are assigned, the formation manager loads from the configuration file the prescribed paths for the flight of the planes to the formation (see the Figure 54). In the formation assemblage procedure, the airplanes come very close one to another - the distances are much smaller than the safety zone of the airplanes. Therefore it is necessary to turn off the collision avoidance algorithm and ensure that the airplanes won't collide. The collision avoidance for the part of the flight from the assemblage start position (ASP) to the formation position (FP) is guaranteed by the prescribed flight paths. These paths are prescribed from ASP to FP. Every position in the formation has its' own flight path. The paths are stored as the flight plans for the formation decomposition and they are reversed using the internal AGENTFLY flight plan utilities. This is the easiest way how to determine the ASPs for different position and orientation of the corridor.

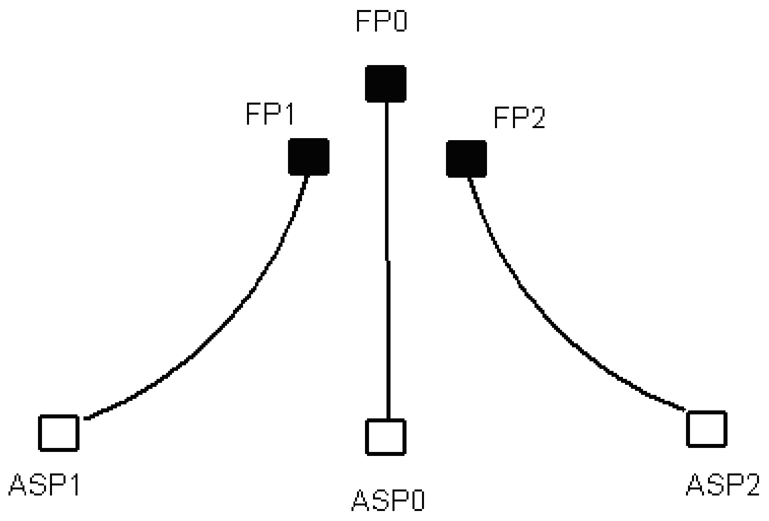


Figure 54: Formation flight-in prescribed paths (for three planes).

As soon as ASPs are determined, the formation manager sends them to the airplanes together with the prescribed flight plan that describes the path from ASP to the FP. The airplanes remove the rest of their actual flight plans (e.g. application of holding orbit) and add the ASP way-point to the end of their flight plan. Their planners plan the path from their current position to assigned ASP point. Since the standard collision avoidance methods have been turned off, it is necessary to ensure that the airplanes don't collide on their way from the current positions to the ASPs.

16.2.3 Collision Avoidance during Approach to ASP

The flight from the current airplane position (CP) to the FP consists of two parts. The first part is the flight from the CP to the ASP, the second part is the flight from the ASP to the FP. The second part is guaranteed by the prescribed flight plans, the remaining task is to ensure the collision avoidance for the flight from the holding orbit to the ASP.

The airplanes send to the formation manager their flight plan, respectively the part of the flight plan that represents the flight from the CP to the ASP. The formation manager performs checks if any two flight plans collide using the similar detection process as in collision avoidance. If the flight plans don't collide, the acknowledge is sent to all formation members and the formation assemblage is agreed and begins. If any two of the flight plans collide, the extra collision avoidance algorithm first tries to change the flight plan for the first airplane. It changes its' flight plan with the flight plan of the aircraft from the same group that has the closest lower angle value. If this change doesn't help and there are still colliding plans, the flight plan of the first airplane is changed with the second lower angle value and the collision is checked again. The algorithm works like this until the changes of the flight plans between the first airplane and all members of its' groups are executed. If the solution is still not found, the same algorithm is executed on the second airplane. When the solution is found, the airplanes start the formation assemblage execution.

16.2.4 Flight in Formation

When the formation is assembled, the airplanes fly in this formation on the same speed from the formation corridor start position to the formation corridor end point. The same speed is guaranteed by the prescribed flight plans - the airplanes fly to the formation on the very same speed and they keep this speed the whole time they are in formation. Current implementation of the formation management in the AGENTFLY supports only formation flight on the straight corridors and no turns of whole formation package are enabled.

16.2.5 Formation Decomposition

When the airplanes reach the formation corridor end position, they follow the prescribed flight plans for the formation decomposition. When the airplanes reach the endpoint of these prescribed plans, the collision avoidance algorithm is turned on again and the airplanes continue in executing their own missions using automatic avoidance mechanism.

17 Experiments

This section summarizes the results of the experiments performed within the project in the AGENTFLY system. First, the scalability tests comparing all implemented cooperative collision avoidance methods are presented in the Section 17.1. The next Section 17.2 presents the comparison of the non-cooperative collision avoidance based on dynamic no-flight zones and the optimal proportional navigation algorithm in several testing configurations. The next set of experiments validating the iterative peer-to-peer algorithm done in the project extension is described in the final report of the project extension [9].

17.1 Cooperative Collision Avoidance

Three implemented cooperative collision avoidance methods were compared in pairs: rule-based vs. iterative peer-to-peer algorithm (Section 17.1.1) and iterative peer-to-peer (IPPCA) vs. multiparty algorithm (MPCA) (Section 17.1.2). Moreover the IPPCA and MPCA were visually evaluated in the specific scenario setup (Section 17.1.3).

17.1.1 Rule-based vs. Iterative Peer-to-peer Collision Avoidance

The results of the comparison of the rule-based (RBCA) (Section 14.4) and iterative peer-to-peer (IPPCA) (Section 14.5) methods are provided here. The IPPCA was limited to apply only changing operators that do not change the altitude of the airplane. The experiments have been carried out within the square area of 31 × 31 `units`. This simulation length `unit` is the same for all experiments.

number of aircraft	5 – 85
each configuration repeated	50 times
testing square area size	31 `units` x 31 `units`
safety range radius of the airplane	0,25 `unit`
radar range radius of the airplane	10 `units`
speed of the airplane	0,075 `unit`$\cdot s^{-1}$ – 0,125 `unit`$\cdot s^{-1}$
aircraft acceleration/deceleration	0,05 `unit`$\cdot s^{-2}$
fastest flight across the square	248 seconds (max. speed on short way)
change point distance	2 seconds from current position
total number of experiments	1700
total number of simulated aircrafts	76490
total flight time	229 hours 44 minutes 34 seconds
total simulation time	54 hours 59 minutes 24 seconds

Table 1: The facts about scalability experiment comparing RBCA and IPPCA using the simulation length unit `unit`; time is measured in the flight time

The Table 17.1.1 summarizes the facts about the experiments. The time in the table is measured in the flight time. The sequence of 850 experiments has been measured for both methods of cooperative collision avoidance. Configurations using 5, 10, 15 ... 85 simultaneous aircrafts flying across the testing area were used. Each experiment was executed 50 times to provide average result values. The size of the on-board radar range radius is 40 times larger than the size of the airplane's safety zone radius. The position of the change point during the negotiation is set to the point which the plane will reach after 2 seconds of the flight time from that moment. Providing that the airplane flies at its maximal velocity and follows a straight flight path, the change point is a point on the edge of the safety zone of this airplane.

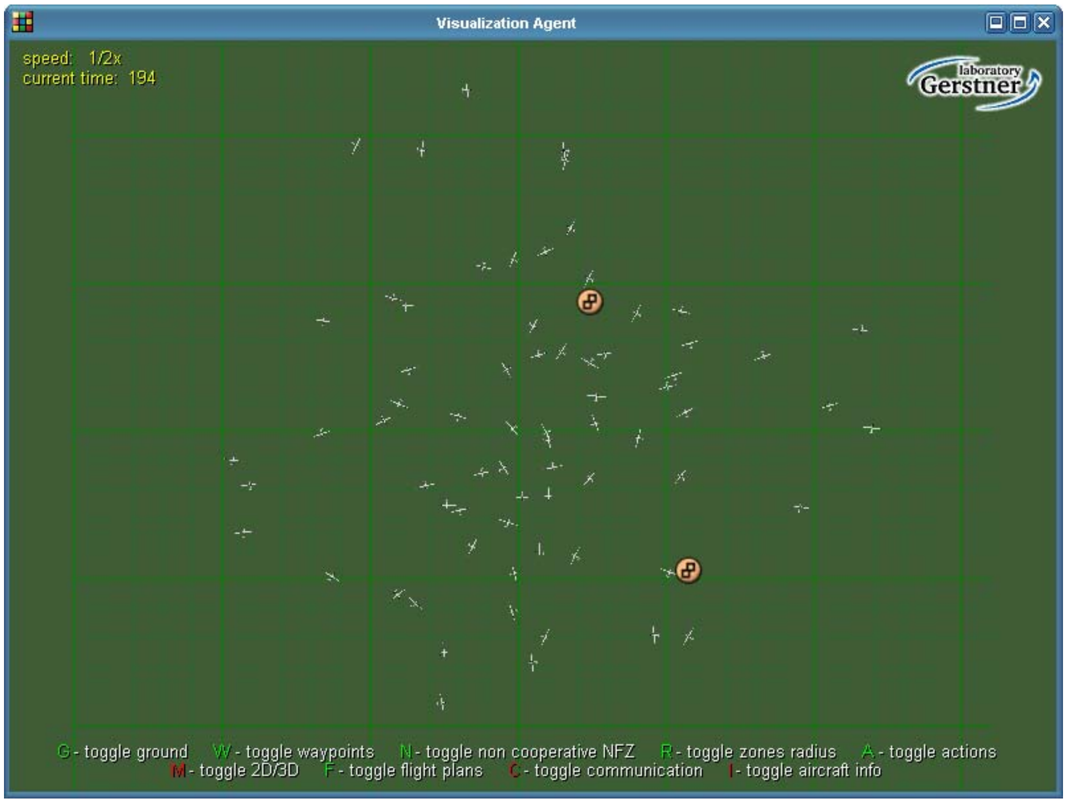


Figure 55: Random generation of the planes for the scalability experiment benchmark

The airplanes were generated randomly in the experiment area as shown in the Figure 55. The worst-case scenario was used. The plane was started randomly on one of the four sides of the experiment area and with the intention to fly across the square to the randomly generated destination point on the opposite square side. There was one restriction for the plane generation: a new plane can be created only if its safety zone is free of any other airplanes. All planes' start and destination points were located on the same flight level. Due to the rule definition used in RBCA (described in the Section 14.4) all aircrafts fly at the same flight level during the whole experiment. Because of this, we omit climb up and descend down operators in IPPCA which leads to flight on one flight level too.

According to the Table 17.1.1, 27010 airplanes were simulated within 1700 experiments, more than 229 hours of flight time were simulated in less than 54 hours of real time.

The Figure 56 presents the comparison of the average number of the safety zone violations. Method based on IPPCA has only a few safety zone violations. In the chart we can observe, that average number is almost zero. On the other hand the RBCA is "much more" worse. And its number of violations is also growing with higher number of planes.

The next Figure 57 shows the average number of flight plan changes. The number of flight plan changes in each experiment is equal to the number of collision avoidances executed by all airplanes in the experiment. Therefore we can simply compute that each plane performs around 4 rule-based and 1.07 pair negotiation about collision in the IPPCA in the configuration with 80 aircrafts.

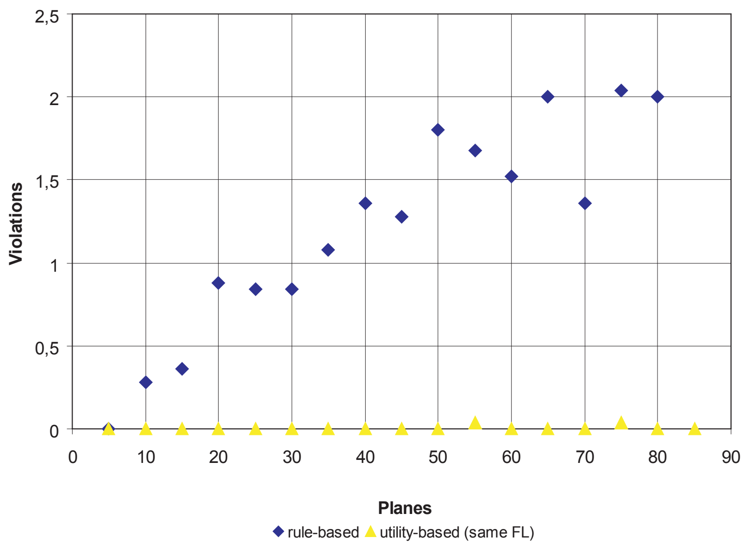


Figure 56: RBCA vs. IPPCA: Average number of safety zone violations

The next chart in the Figure 58 shows the average sum of differences between the final collision-free (non-colliding) and the initial flight plan for all planes in the specific experiment configuration. For the configuration with 80 planes in the experiment the average increment of the flight plan length was 9 `units` for RBCA and almost 0 for IPPCA. It means that each plane had to increase the length of its flight path by about 0,11 `units` in scenario with RBCA which represents less than 0.4 percent difference from the shortest (original) path. For the scenario with IPPCA it is even less. In the Figure 59 you can see flight plans of planes in scenario with RBCA. The Figure 60 shows flight plans of planes in scenario with IPPCA. It is nice demonstration how it is possible that there can be so big difference between the results from these two methods of collision avoidance. The doted lines in screen shots represent original straight flight plans of UAAs. And solid lines represents the final flight plans. You can see that in the scenario with RBCA are lines of current flight plans much more curved.

Even if we omit operators turn down and turn up in IPPCA, the planes have one big advantage with this method. They can slow down or speed up a little. With this advantage they can keep almost straight flight plan, because they can solve collision by acceleration or deceleration giving the best value for the utility function.

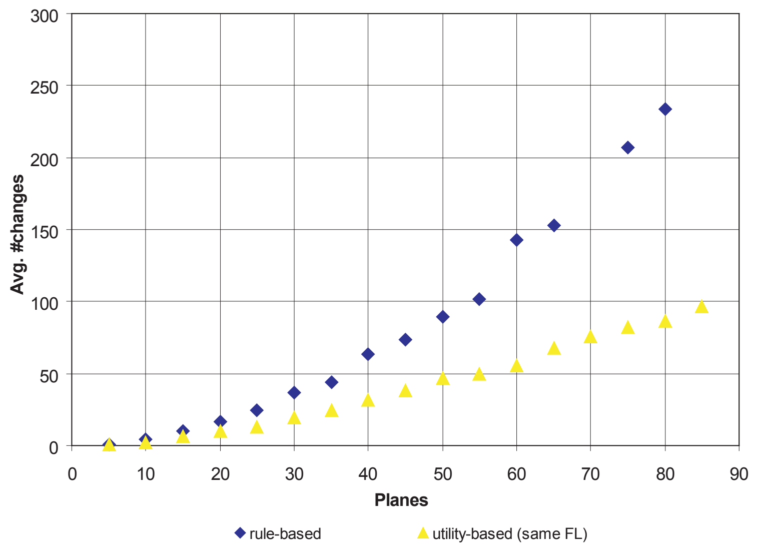


Figure 57: RBCA vs. IPPCA: Average number of changes in flight plans

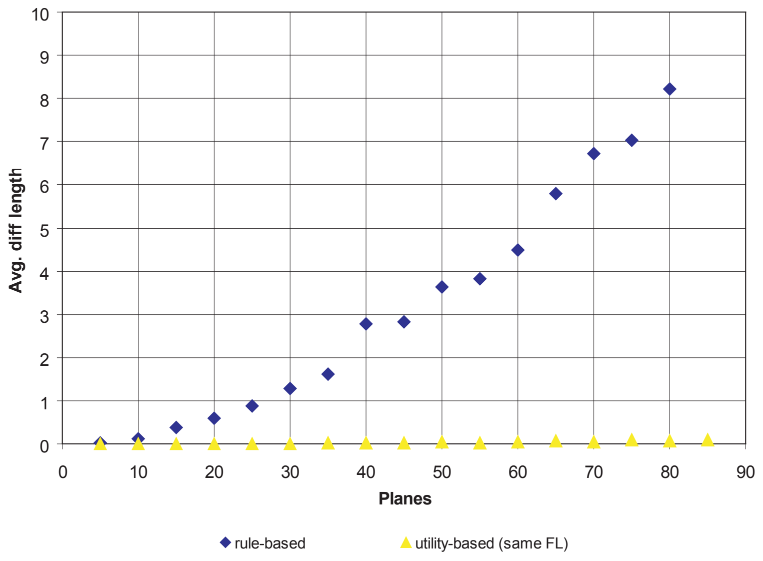


Figure 58: RBCA vs. IPPCA: Average diff length of final flight plan vs. initial straight flight plan

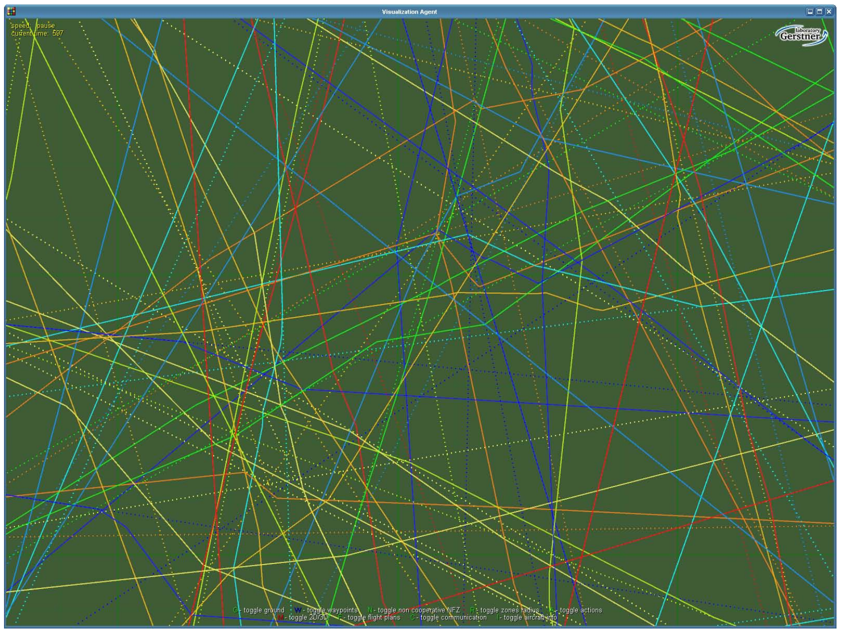


Figure 59: RBCA vs. IPPCA: Original flight plan (doted line) and final flight plan (solid line) in scenario with rule-based deconfliction

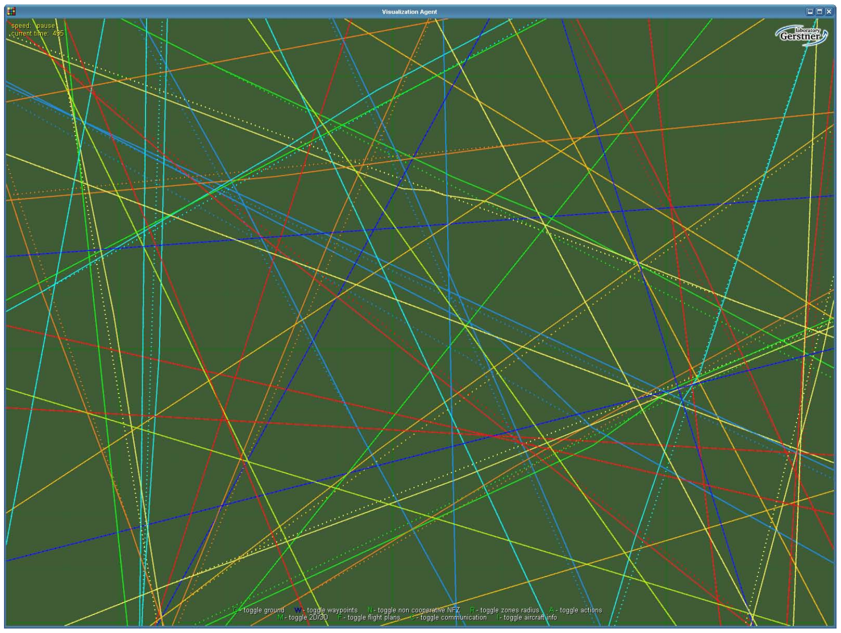


Figure 60: RBCA vs. IPPCA: Original flight plan (doted line) and final flight plan (solid line) in scenario with utility-based deconfliction

The next chart 61 shows the performance of the designed AGENTFLY prototype. The experiments were started in the fast simulation mode. In this mode the real-time visualization components were disabled and the automatic adaptive simulation speed was used. The speed of the simulation (the ratio between the simulated flight time and the real time) is dynamically regulated depending on the overall system load.

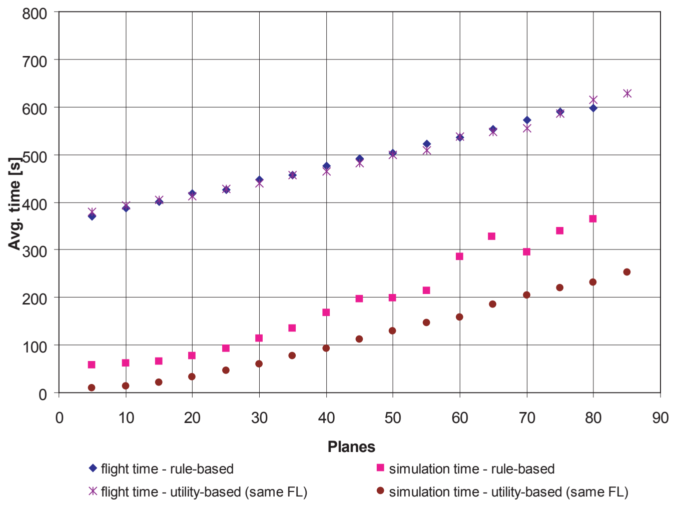


Figure 61: RBCA vs. IPPCA: Comparison of the experiment flight time to simulation time

The next chart 62 shows the average communication data flow among all airplanes. It corresponds to the sum of a size of all IP packets used during communications among UAAs. It is interesting that even if the average number of flight plan changes is higher for RBCA, the average data flow is almost the same in both cases. This nicely indicates a need of higher amount of data for solving one collision in IPPCA.

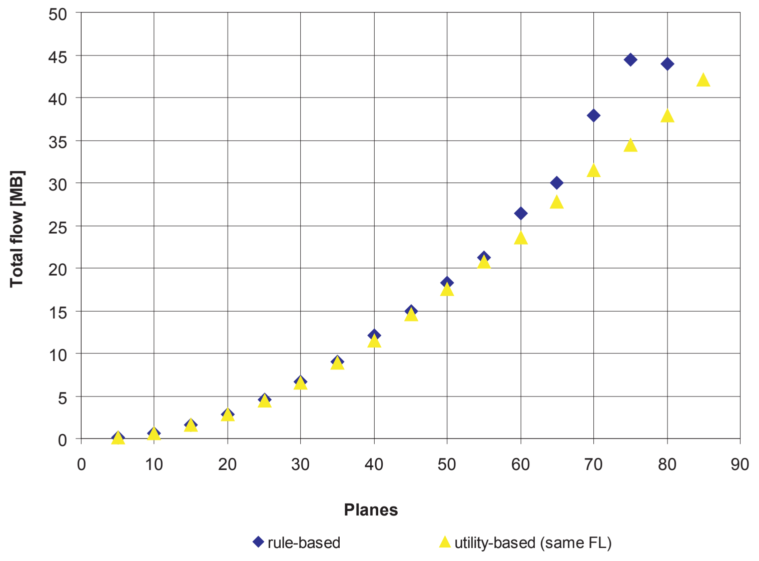


Figure 62: RBCA vs. IPPCA: Average communication data flow among all UAAs

The next chart 62 contains a maximum data flow between any two UAAs in each second. The data are from configuration with 70 planes. For each second there is one dot. This makes the dots in some places really concentrated. It shows three main levels in the maximum data flow. And we can see that a single plane has highest demands on the communication at the first half of the simulation.

The next chart 62 shows minimal separation between UAAs in each second of simulation. The red solid lines in the chart show the size of the safety zone around each UAA. The simulation is for 70 planes. We can see that after a short time the minimal separation falls near the safety range of planes.

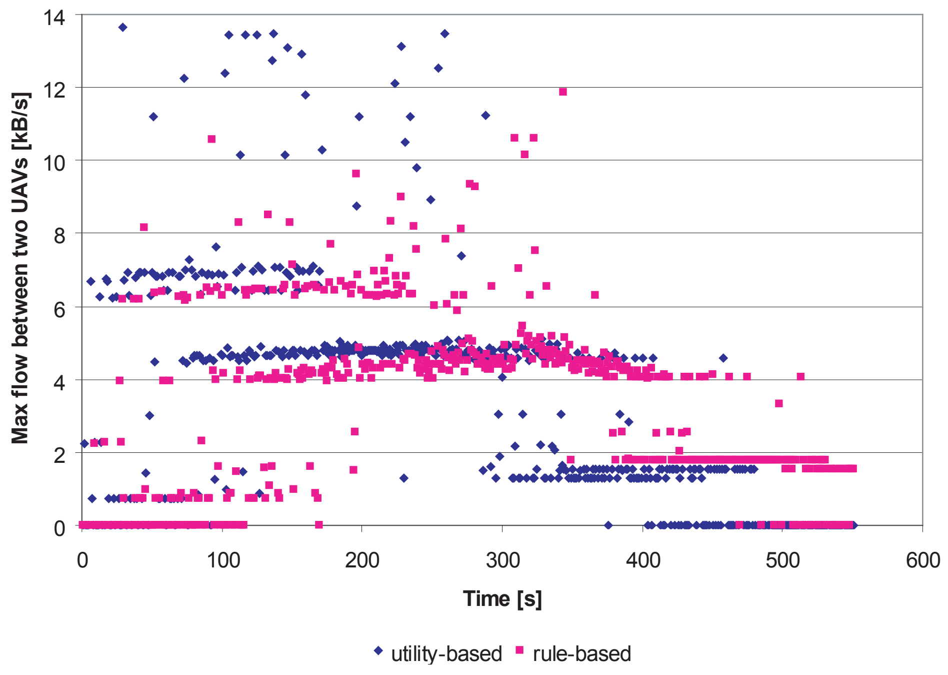


Figure 63: RBCA vs. IPPCA: The maximum data flow between any two UAAs in scenario with 70 planes

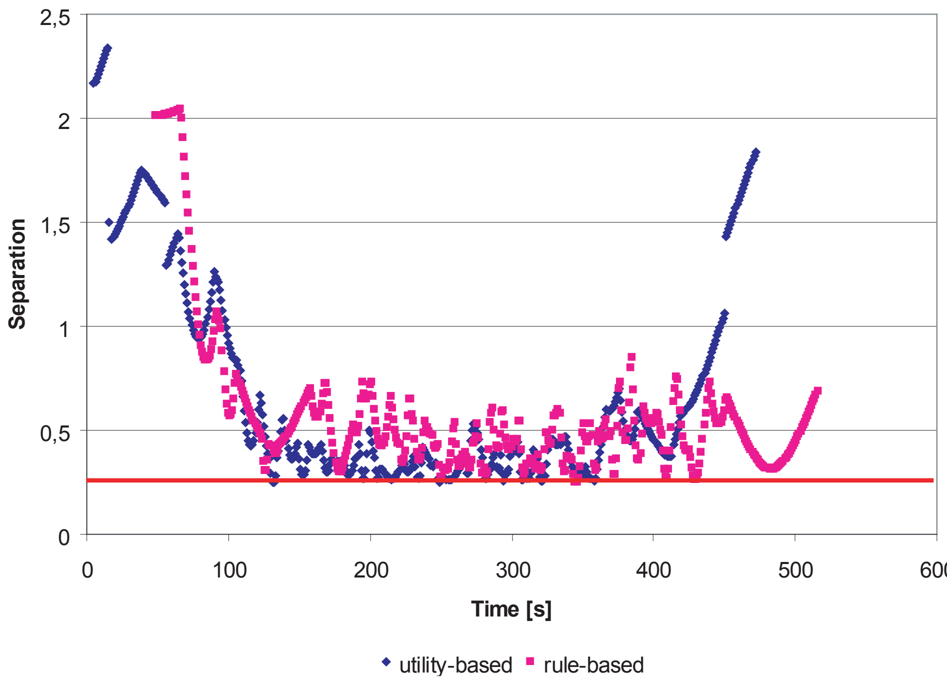


Figure 64: RBCA vs. IPPCA: The minimal separation between UAAs in scenario with 70 planes

17.1.2 Iterative Peer-to-peer vs. Multi-party Collision Avoidance

In the random experiment we performed a set of repetitive tests while collecting several characteristic properties for the comparison of iterative peer-to-peer (IPPCA) (Section 14.5) and multi-party collision avoidance (MPCA) (Section 14.6) methods. The sequence of 900 runs (configuration with 5, 10 ... 90 airplanes each 50 times repetitively) for each method was carried out in the limited airspace area of 31 x 31 units. The mission M_i holds exactly two way-points randomly generated on the two opposite airspace borders, thus each airplane needs to fly across the square. All way-points have the same altitude during whole experiment and each new airplane's way-points are generated on the adjacent borders of the square in clock-wise direction. Such generating scheme guarantees high number of collisions in the middle of the airspace. The safety zone size $rsz = 0.25$ units and the communication range $c = 10$ units are the same for all airplanes. The airplane flying speed can vary between 0.075 and 0.125 units per second. Totaly 85,500 airplanes were simulated during more than 230 hours of flight time. The collision-free solution was found in every simulation run – there is no collision between any two final airplanes' flight plans.

The random generation is the same as in the comparison of RBCA and IPPCA (Figure 55). The radius of each UAA's safety zone is 0.25 units and its radar range radius is 10 units. The UAA's flight speed can vary between 0.075 and 0.125 units per second with acceleration and decelaration of 0.05 speed units per second. Totaly 85500 UAAs were simulated during more than 230 hours of the flight time. A collision-free solution was found in each run – all UAAs respect the safety area around all others during the simulation.

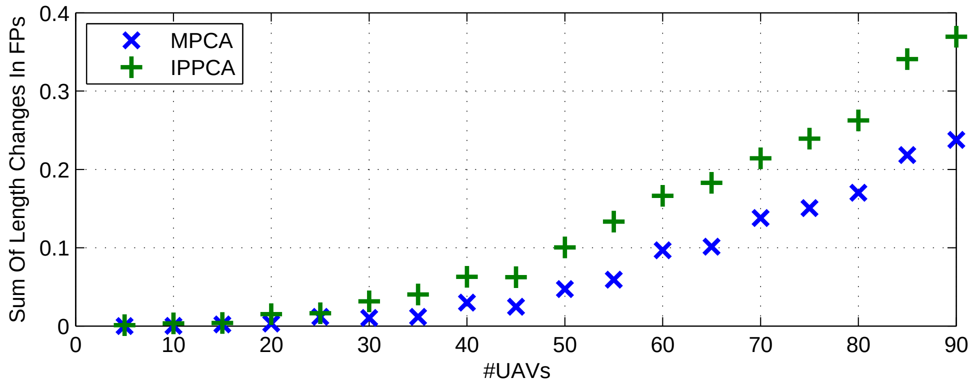


Figure 65: IPPCA vs. MPCA: The sum of differences between final collision-free paths and shortest regardless collisions.

The top plot in the Figure 65 presents the comparison of the average sum of all differences between the final collision-free flight plans and the shortest path from start to end way-points (euclidian distance) in given run. The results validate benefits of the MPCA algorithm to provide a more optimal solution – depending on the number of UAAs, the results are improved by 10 to 50 percents compared to the IPPCA. The value varies due to the fact that the same numbers and types of collisions during each randomized experiment are not guaranteed.

The chart in the Figure 66 displays the average sum of numbers of changes applied by all UAAs in the given experiment run. More changes occur in the IPPCA due to its iterative nature of solving multi-collisions – the multi-collision is a collision of more than two UAAs at the same place and time. The difference in the number of applied changes in a flight plan correlates with the difference depicted in the previous plot. There are situations where multi-party coordination group has more than two UAAs that participate in searching a solution.

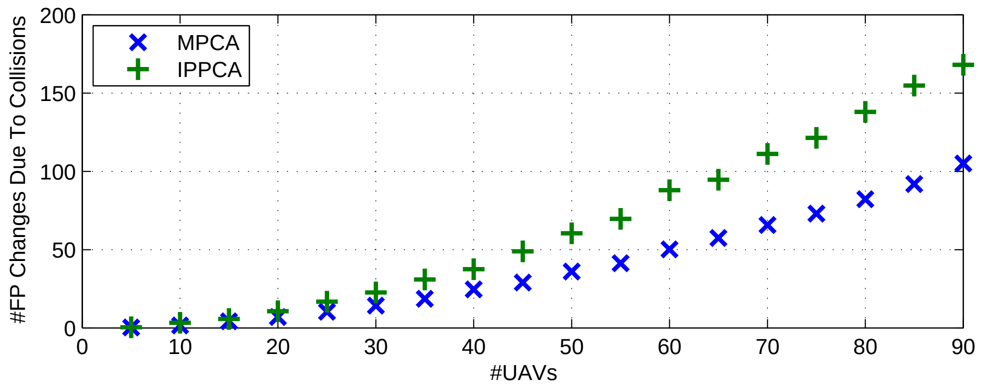


Figure 66: IPPCA vs. MPCA: The number of applied changes by all planes in particular experiment run.

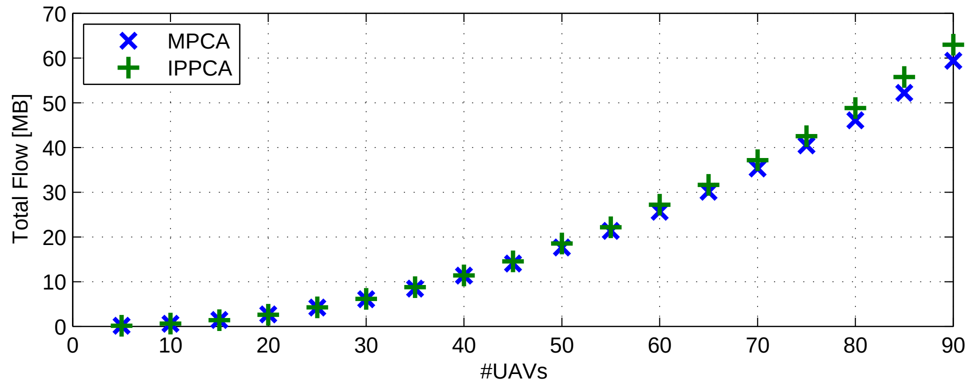


Figure 67: IPPCA vs. MPCA: The communication flow analysis: total communication flow

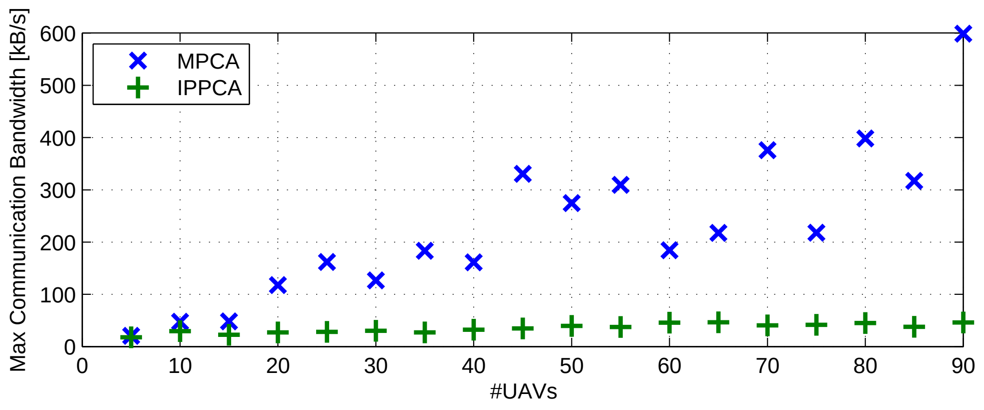


Figure 68: IPPCA vs. MPCA: The communication flow analysis: maximum network bandwidth (middle)

We've studied communication aspects of both algorithms, see Figure 67, 68 and 69. Both algorithms have almost the same amount of transmitted bytes, but the difference is in the flow distribution during the experiment run. The middle chart presents the maximum communication bandwidth depending on the number of UAAs in the experiment run. Again the chart presents the average value derived from **50 repeated** tests. We observed that the MPCA requires several times wider communication bandwidth than IPPCA, especially for the runs with more UAAs. The bottom plot shows the communication flow over the time. The MPCA requires more communication during the state expansion phase within the coordination group but on the other hand it requires smaller number of coordination groups due to the fact that MPCA solves multi-collision in one change. This leads to almost the same sum of total bytes transmitted among all UAAs in the run.

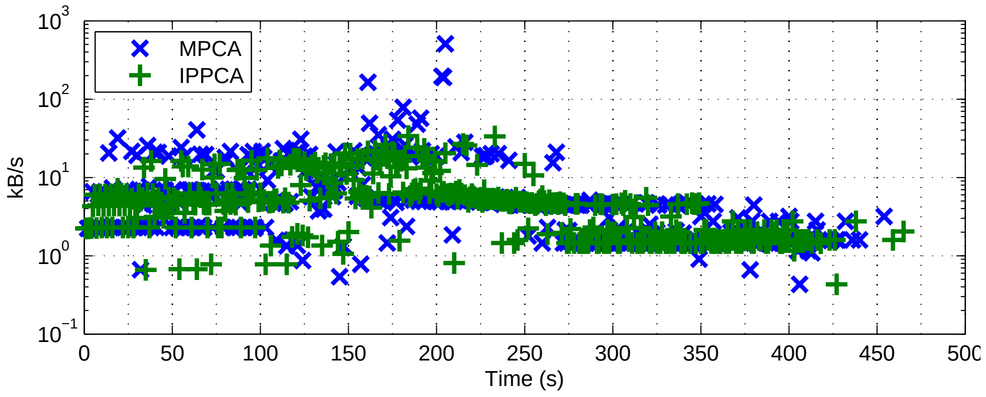


Figure 69: IPPCA vs. MPCA: The communication flow analysis: network flow distribution in time in one specific run for 90 UAAs (bottom).

We've also analyzed the computation requirements of both algorithms for the entire runs. The computation requirements is measured as a sum of time necessary to solve all collisions in the run. Both experiments have been carried out on the same identical computer. We found that the requirements are also almost the same for both algorithms, but the distribution of computation power is similar to the distribution difference in the network flow.

17.1.3 Iterative peer-to-peer and Multi-party Method in Specific Scenarios

The difference between IPPCA and MPCA algorithms have been tested on two selected worse-case based scenario setups. In the first setup there are flying 13 airplanes located in the geometrical vertically oriented plane. Their position in the plane is shown in the Figure 70 left. Initially all of them fly in the same direction and at the same flight speed. There is another airplane which is flies in the opposite direction and has a head collision with the airplane located in the middle of the first group. The final results comparing both collision avoidance methods are depicted on the right side in the Figure 70. When using the iterative peer-to-peer algorithm only one plane avoids the group of airplanes and therefore no other planes participate in the solution. On the other hand, while using the multi-party method the middle airplane in the group performs a combination of several flight plan changing manoeuvres and creates a small hole in the middle of the group flying in geometrical plane to let the opposite airplane fly through. Then the airplane goes back to its original central position within the group. The difference is given by the fact that the multi-party solution is found by the search for the solution with best criterion through the large space of possible combinations. In iterative peer-to-peer version two negotiating airplanes must find solution of their collision and next negotiation cannot take already applied changes back. The multi-party method provides a solution that is only 0.213 units longer than the initial flight plan while the iterative peer-to-peer gives a solution that is 2.843 units longer. These values were calculated as an average from 20 consecutive experiments.

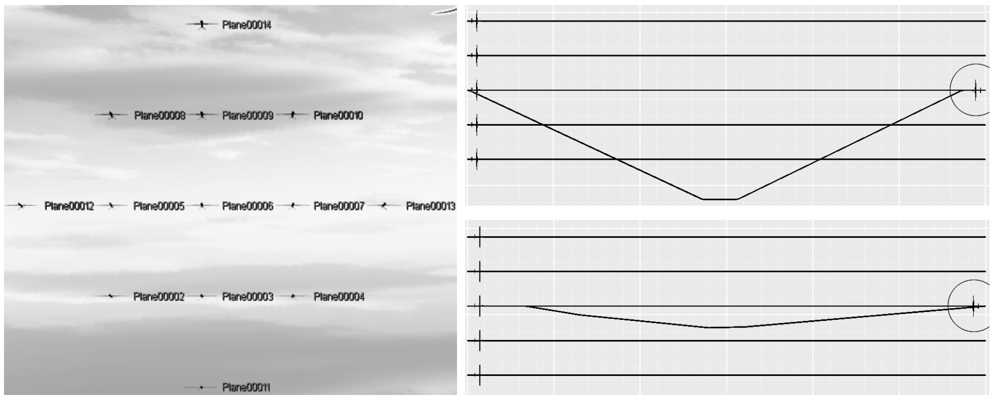


Figure 70: Left: the setup of the first test scenario. Right: the result after iterative peer-to-peer (top) and multi-party (bottom) negotiations.

The second selected worse-case based scenario setup places ten airplanes equally to the horizontal circle. All of them start at the same flight altitude and want to fly to the opposite side of the circle through its center. Therefore in the center of the circle there is a multi-collision of all airplanes where each one has collision with all others. Both algorithms find a solution for the setup – final flight plans are not colliding together. The iterative peer-to-peer algorithm produces final flight plans 1.963 units longer then the initial ones and the multi-party method provides solution only 1.441 units longer.

17.2 Non-cooperative Collision Avoidance

The functionality of the implemented algorithms of the non-cooperative collision avoidance (see 15) was verified using two sets of experiments. In the Section 17.2.1 there are experiments comparing the no-flight-zones-based non-cooperative collision avoidance (NFZCA) (described in the section 15.1) with the optimal proportional navigation (described in the Section 15.2). Additional scenario where proportional navigation fails is presented in the section 17.2.2. The Section 17.2.3 presents tests with complex scenarios using the NFZ-based non-cooperative collision avoidance only.

17.2.1 Comparison of No-flight-zones-based Non-Cooperative Deconfliction and Proportional Navigation

Comparison scenarios are based on the situations described directly in [4]. There are three of them, each contains one controlled airplane and one uncontrolled. The uncontrolled airplane (called an obstacle) always flies directly from the starting point to the destination point. The controlled airplane is always heading north and it must avoid the collision in the middle of the operation area.

As the scenarios are supposed to be planar, it was necessary to disable changes of altitude in case of non-cooperative collision avoidance. This was simply achieved by removing the spatial search manoeuvres for climbing and descending from the flight path planning algorithm (see the Section 6.1).

Table 2 shows the comparison of lengths of average trajectories (10 measurements) using proportional navigation and NFZ-based non-cooperative collision avoidance.

	average trajectory length $l[u]$		
	proportional navigation	NFZ-based non-cooperative CA	no control
perpendicular collision	33,21	33,85	30,0
slant collision	31,17	31,52	30,0
head-up collision	30,81	30,62	30,0

Table 2: Comparison of lengths of average trajectories using the proportional navigation and NFZ-based non-cooperative CA

I. Perpendicular Collision Scenario

The first comparison scenario involves the perpendicular collision. The resulting flight trajectories can be seen in the Figure 71. The left trajectory is the result of the proportional navigation and the right one is the result of NFZCA. The left trajectory is smoother, because the proportional navigation uses calculation of acceleration vector **a** in each step of the simulation. The red color corresponds to the avoidance manoeuvres and the green color marks the flight towards the destination point. In the right trajectory we can see search manoeuvres of manoeuvre-based spatial planning (see 6.1) used by NFZCA.

The length of the safety zone towards an uncontrolled airplane is depicted in Figure 72 using a line of the given length pointing towards the uncontrolled airplane.

The predicted no-flight zone of a non-cooperative object during a simulation run can be seen in Figure 73. In this case the zone is not deformed in any way as the figure captured the particular moment when replanning occurred and in such a case the zone is normalized to the unit length.

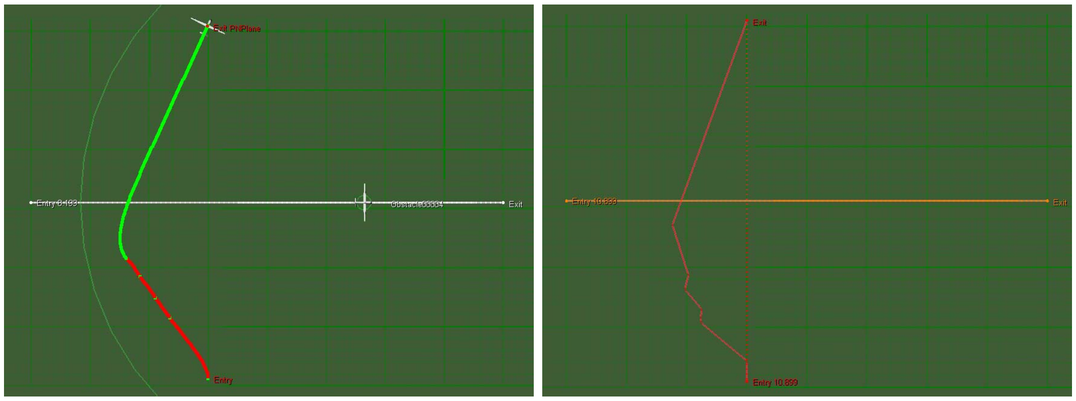


Figure 71: Results of the perpendicular collision scenario

Chart 74 shows relation between the distance of a controlled and uncontrolled airplane and time. In contrast to NFZCA, the proportional navigation plans the trajectory so that it touches the edge of the safety zone. NFZCA leaves greater distance between the trajectory and the object, because a dynamic no-flight zone also considers the possible change of direction of the object.

II. Slant Collision Scenario

In the scenario of slant collision (Figure 75) we can see the deviation from the nominal trajectory occurred later and is less prominent than in the previous case.

Similarly to the previous case, the chart 76 shows a comparison of distances between controlled and uncontrolled airplane for both algorithms.

III. Head-up Collision Scenario

The last comparison scenario (Figure 77) is specific for its singular configuration of starting and destination points. In the case of proportional navigation the problem is solved by directing the airplane so that it follows the tangent of the safety zone circle and the airplane (vector $\overrightarrow{\mathbf{XA}}$). In case of NFZCA the solution depends on the ordering of turning manoeuvres. In this particular case the first one is a left turn.

In this scenario the safety zone is clearly breached (see diagram 78). In case of the proportional navigation the problem is caused by restriction of the maximal acceleration of the airplane, while in case of NFZCA it is caused by the deformation of the dynamic no-flight zone.

17.2.2 Additional Scenario of Proportional Navigation Failure

An additional comparison scenario is a demonstration of restriction and failure of proportional navigation (Figure 79). This simple scenario contains two uncontrolled airplanes configured in such a way that once the algorithm switches to the proportional collision avoidance mode towards the other airplane (Obstacle00004) the conditions for flight-to-destination mode are fulfilled which causes the actual avoidance to be switched off and the airplanes collide.

NFZCA works correctly in this situation, because the planning algorithm considers all current non-flight zones (Figure 80).

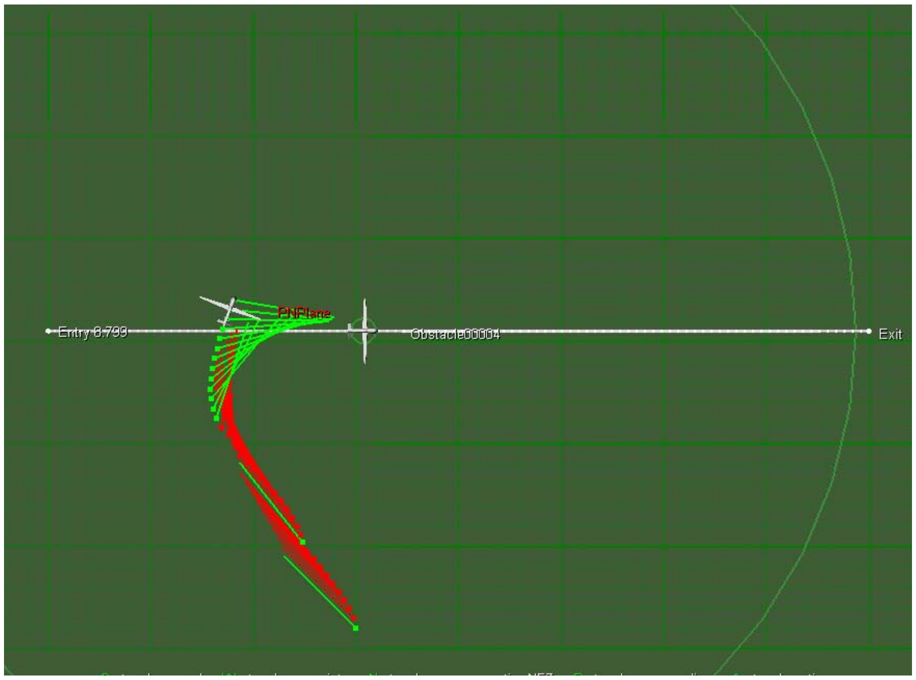


Figure 72: The length of the safety zone during the simulation run (proportional navigation)

The given situation can be compared also in the diagram of the distance of controlled and uncontrolled airplane 81. Proportional navigation algorithm no longer tries to avoid the other object after the first overshoot (at 10 sec).

17.2.3 Complex Scenarios of Non-Cooperative Collision Avoidance

The section presents a set of experiments with scenarios designed specifically for NFZCA. As we see in the scenario of proportional navigation failure (see 17.2.2), no more than two airplanes are sufficient to prove that in case of a suitable configuration, airplanes controlled by a proportional navigation algorithm will crash. Since the following scenarios contains more than two controlled airplanes, the algorithms are incomparable.

In this set it is also necessary that the non-cooperative collision avoidance involves also the altitude changes, because the solution can usually not be found using the planar collision avoidance only.

Table 3 shows comparison of lengths of average trajectories (20 measurements) in three complex non-cooperative collision avoidance scenarios.

	length of trajectory $l[u]$			
	average	minimum	maximum	no control
two airplanes collision	31,58	30,28	37,16	30,0
four airplanes collision	32,29	30,15	42,61	30,0
ten airplanes collision	36,89	30,10	65,49	30,0

Table 3: Comparison of lengths of trajectories in NFZCA scenarios

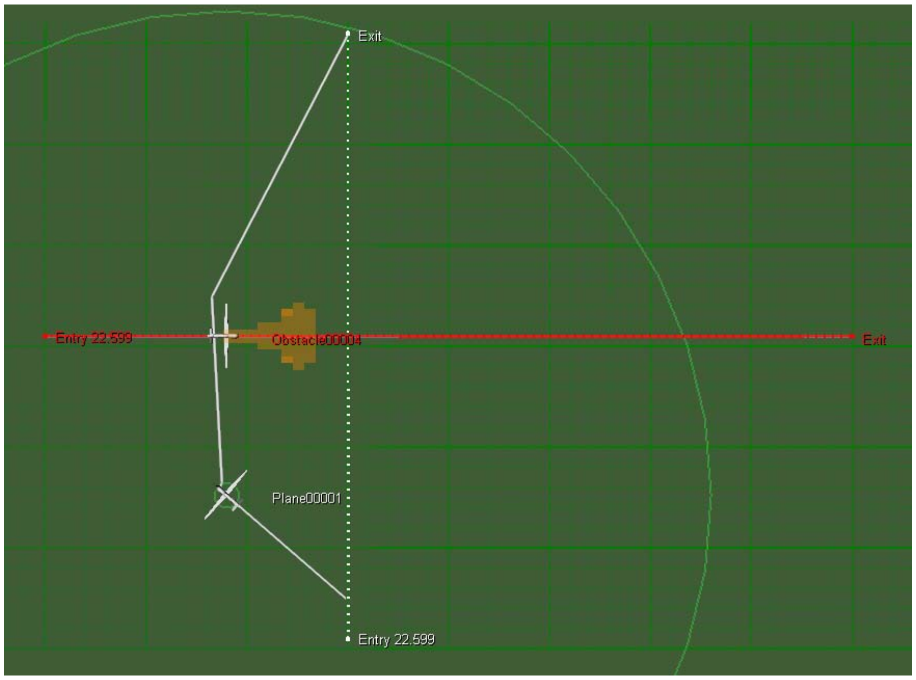


Figure 73: Predicted no-flight zone during the simulation run (NFZ-based non-cooperative deconfliction)

I. Two Airplanes Collision Scenario

The first scenario is a straight head-up flight of two airplanes controlled by NFZCA. Both airplanes predict a dynamic no-flight zone of the other airplane and plan a path around it. In Figure 82 we can see one no-flight zone predicted by airplane Plane00001.

In Figure 83 we can see three different solutions of the given scenario. The non-deterministic nature of the non-cooperative collision avoidance algorithm is caused by the implementation of the pilot agents running in separate threads in the operating system.

Execution and switching between these threads depends on a number of factors that cannot be controlled by the AGENTFLY system (e.g. other running programs, load and number of processors etc.) Threads switching influences ordering of execution of planning algorithms and thus the resulting trajectories as well.

The first trajectory is a result of a planning process that occurs after a no-flight zone is predicted for the first time, i.e. both airplanes turn left. The second trajectory is a result of repetitive execution of the replanning process that is active until it is more beneficial for one of the airplanes to change altitude temporarily and thus avoid the collision (shorter resulting trajectory). The last trajectory is a result of cyclic changes of trajectory applied by both airplanes symmetrically (the problem of singular situation). This problem is solved by a multiplicative constant from the interval $\langle 1; 1,5 \rangle$ generated randomly for each airplane in the system. This constant is used for scaling all predicted zones of the given airplane. This way the probability of collision during the cyclic path changes is remarkably reduced, because the generated paths are different for differently scaled zones.

II. Four Airplanes Collision Scenario

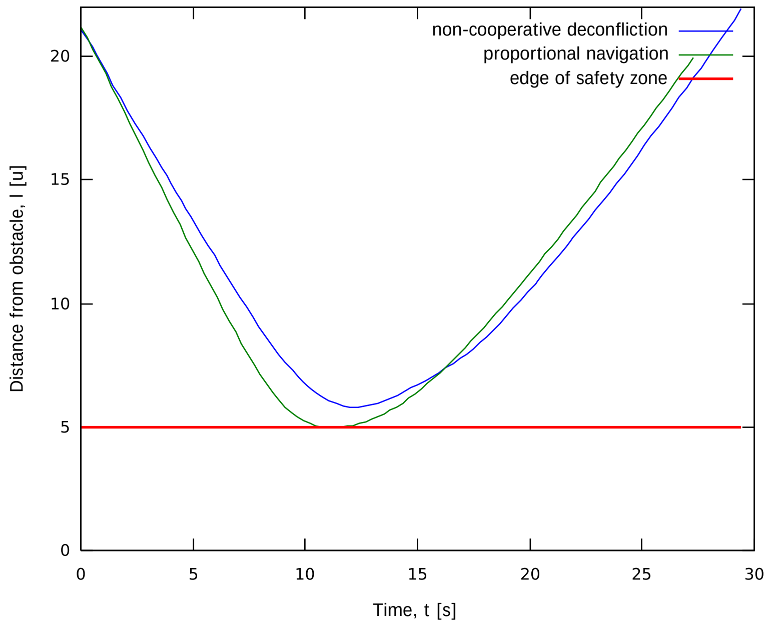


Figure 74: Graph of the distance of the controlled and uncontrolled airplane in the perpendicular collision scenario

By extending the previous scenario with two airplanes flying in the perpendicular directions we get a scenario with four airplanes. In Figure 84 we can see the use of altitude change by the non-cooperative algorithm and on the left-hand side there is a spiral manoeuvre for an "abrupt" change of altitude.

In the Figure 85 we can see the execution of the scenario with two currently predicted zones and two results of the scenario. In most cases one or two airplanes change their altitude during the very first planning process, because the no-flight zones of the perpendicularly flying airplanes form a barrier that cannot be avoided by turning manoeuvres only. As soon as these airplanes start climbing, the collision predictor of the other pair no longer takes the two climbing airplanes into consideration and the collision is solved the very same way as in the previous scenario, two separate instances in two different flight levels.

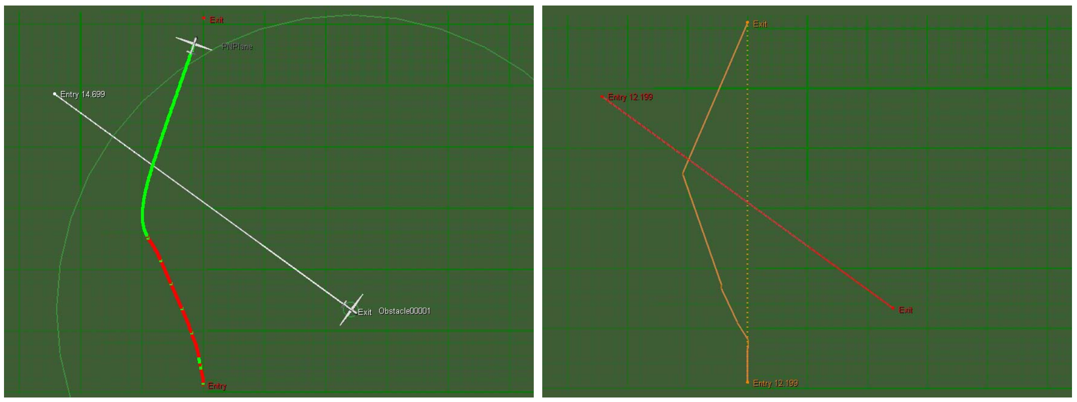


Figure 75: Results of the slant collision scenario

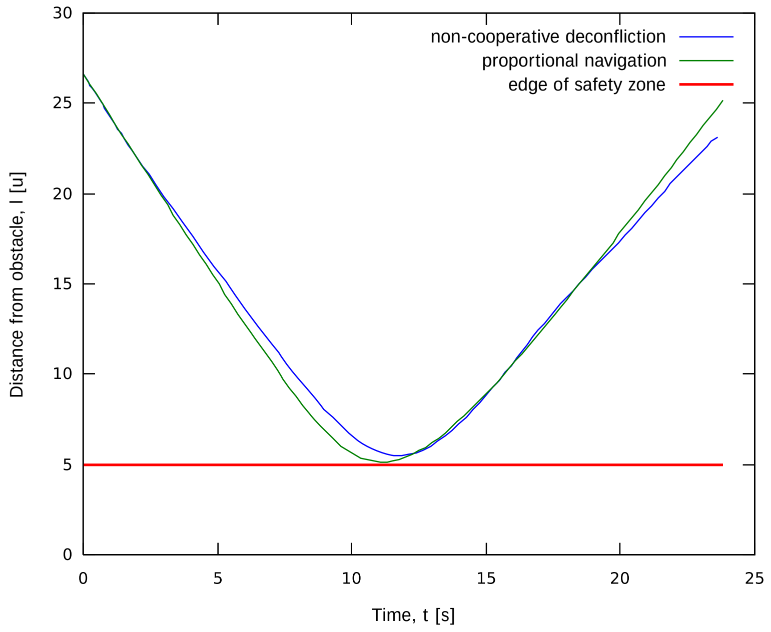


Figure 76: Diagram of distance of controlled and uncontrolled airplane in slant collision scenario

III. Ten Airplanes Collision Scenario

An extreme testing scenario is the collision of ten airplanes with one theoretical collision point in the middle of the operation area. Similarly to the previous scenario the result of the planning process is demonstrated in both 3D (Figure 86) and 2D (Figure 87).

Using such scenario in the non-cooperative collision avoidance setup is unfitting considering the requirements it poses on the collision avoidance algorithm. Nevertheless, in approximately 85% of simulation runs no collision occurs.

It is important to point out that the simulated situation is highly unlikely to happen in real-world cases, moreover the space for manoeuvres is very restricted, only the basic linear collision predictor is used, turn radius is relatively big, considering the size of the operation area and the

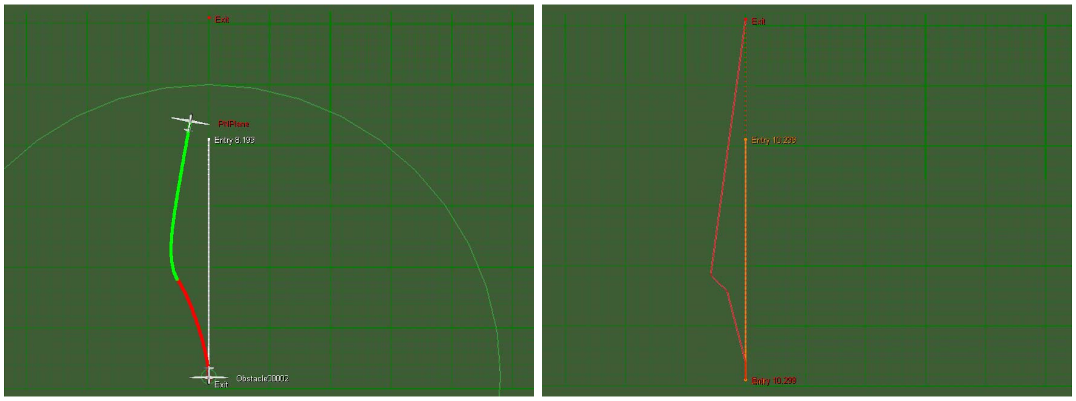


Figure 77: Results of the head-up collision scenario

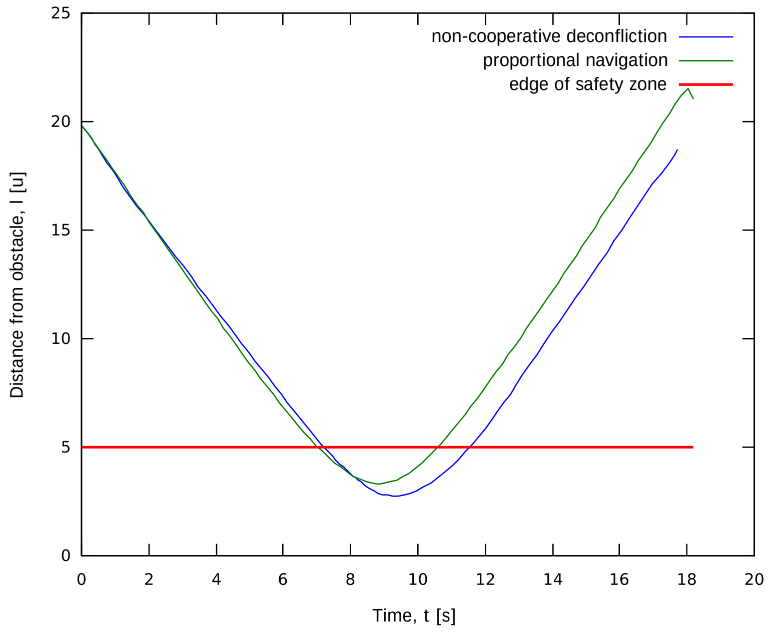


Figure 78: Diagram of distance of controlled and uncontrolled airplane in the head-up collision scenario

given situation and the entire simulation runs on a single processor. Under these circumstances, the success rate of 85% is a very good result.

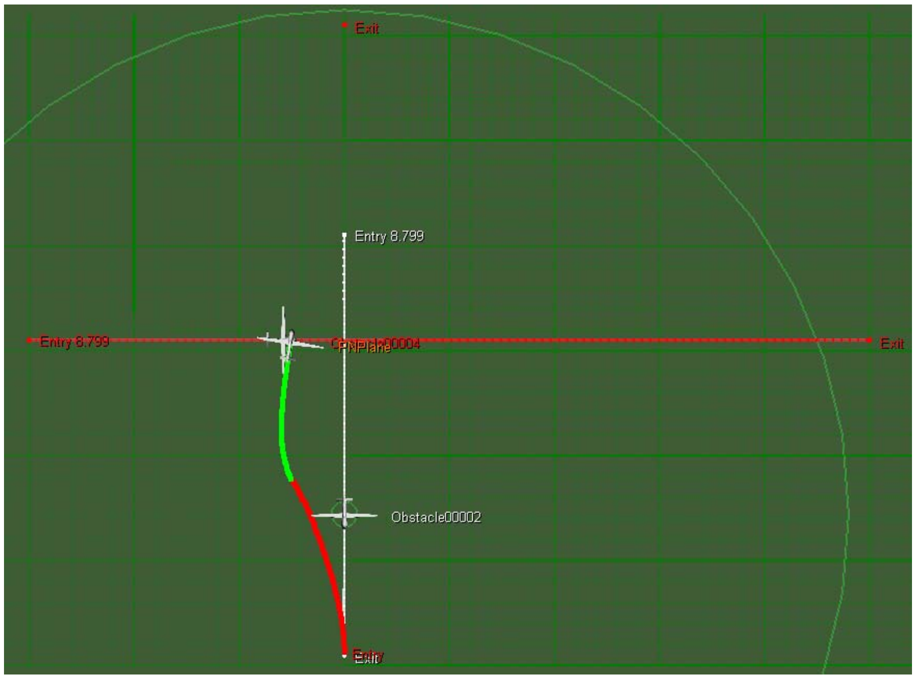


Figure 79: Results of the proportional navigation failure scenario

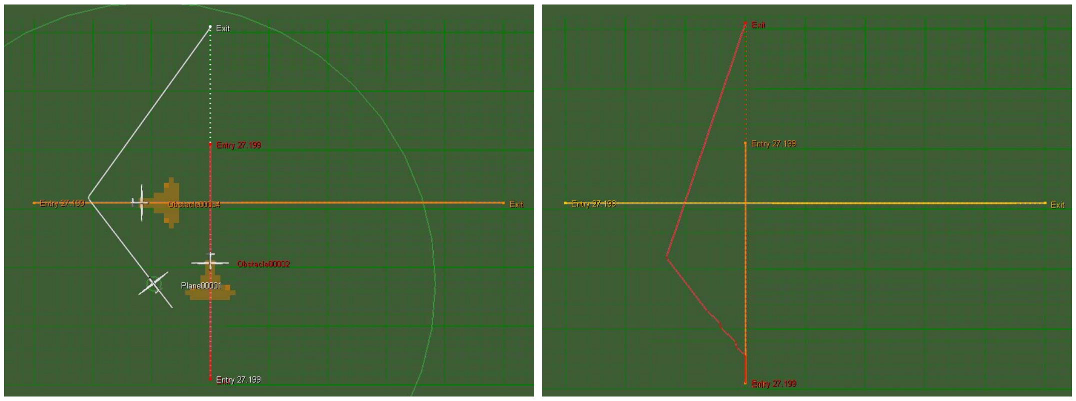


Figure 80: Results of the proportional navigation failure scenario handled by the NFZCA

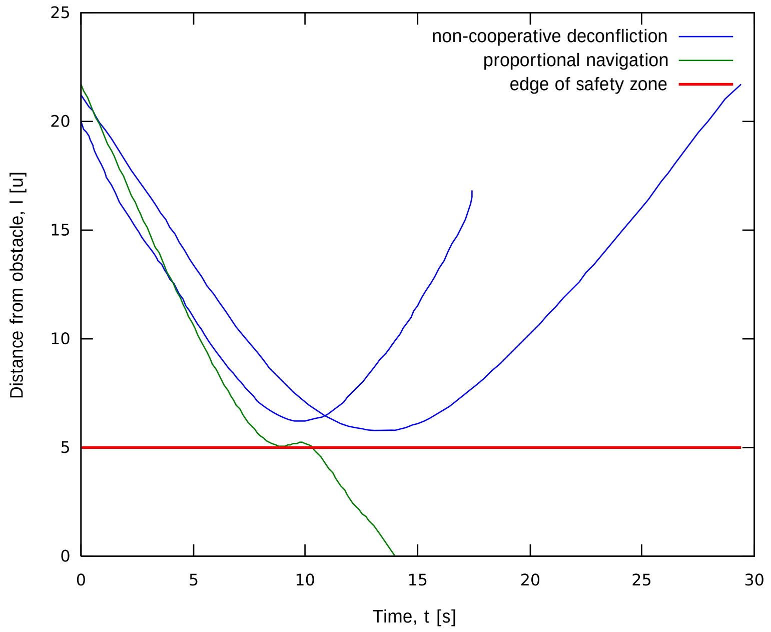


Figure 81: Diagram of distance of controlled and uncontrolled airplane in the proportional navigation failure scenario

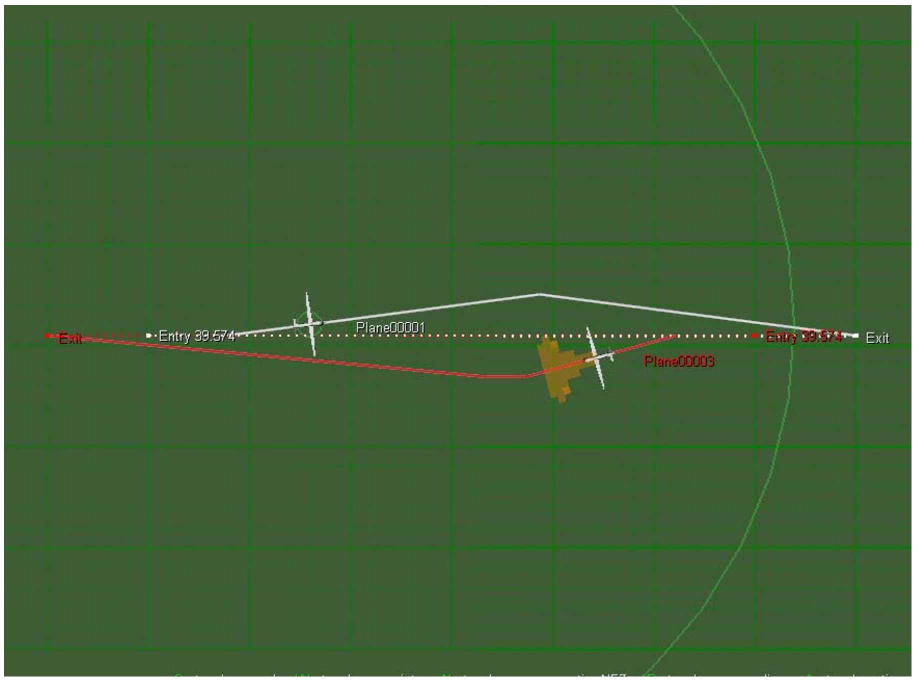


Figure 82: Two airplanes collision scenario

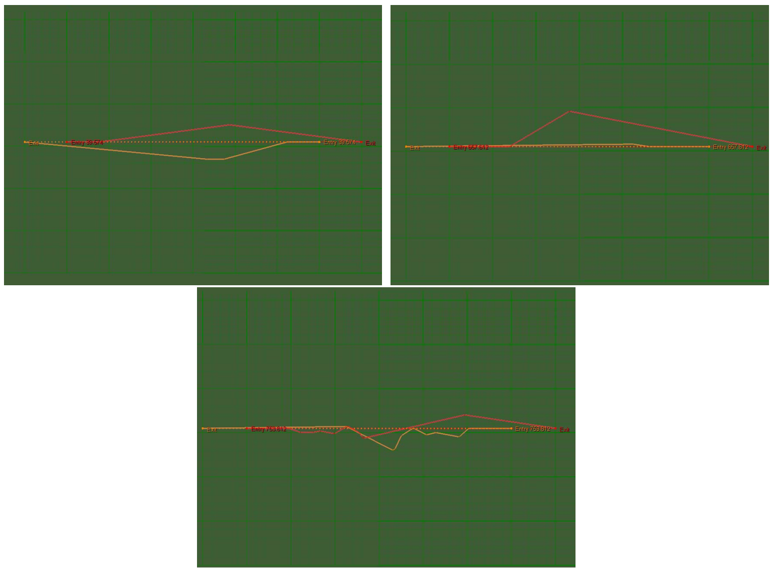


Figure 83: Results of the two airplanes collision scenario

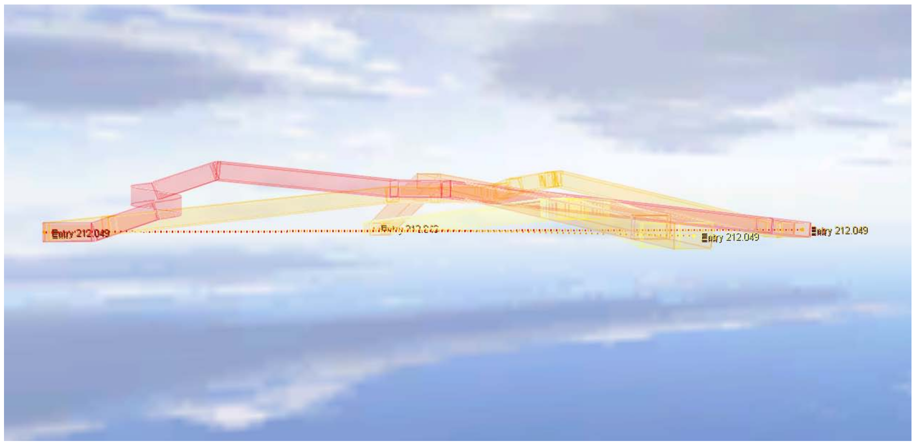


Figure 84: Horizontal view of the result of four airplanes collision scenario

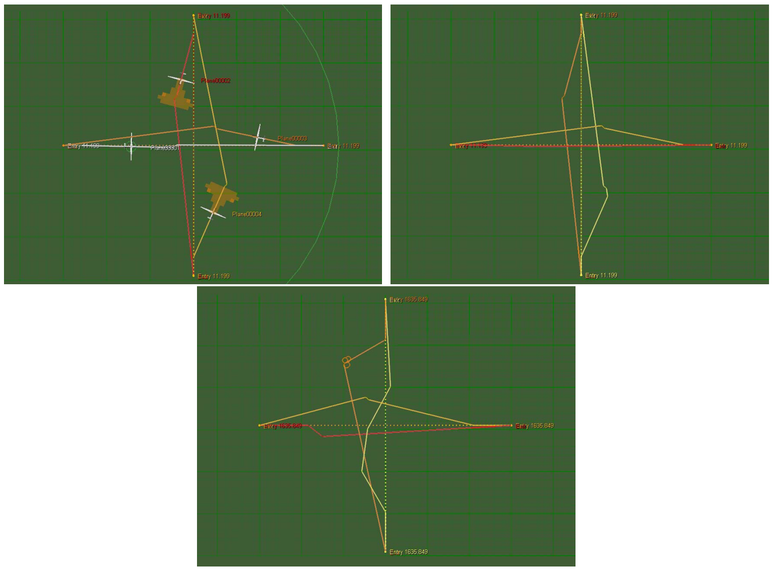


Figure 85: Course and results of four airplanes collision scenario

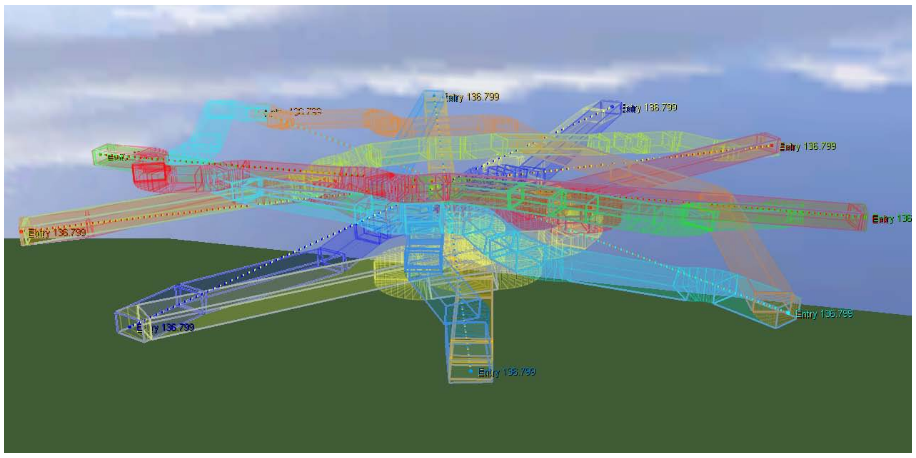


Figure 86: 3D view of the result of ten airplanes collision scenario

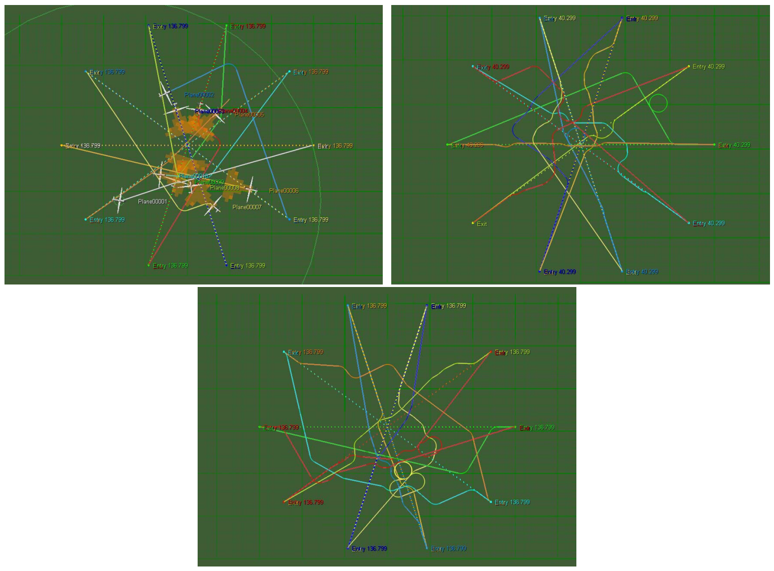


Figure 87: Course and results of ten airplanes collision scenario

18 UAA Operation over LA in Real Civil Air-Traffic

The multi-layer collision avoidance architecture (Section 13) has been validated in the environment where operate real air-traffic over Los Angeles International Airport. In the scenario there are inserted two types of planes. There randomly operate agent controlled UAAs. The UAAs are configured to use the iterative peer-to-peer collision avoidance (Section 14.5) with other UAAs controlled by agents and if there is other flying object they will use the no-flight zones based non-cooperative collision avoidance method (Section 15.1).

The simulated air-traffic is extended by the real air-traffic data obtained from publicly available internet sources, as described in the section 8. These airplanes are detected by on-board radars of the simulated airplanes in the NFZ-based non-cooperative solver and thus the collision avoidance loop (collision prediction, dynamic no-flight zones and planning) is triggered. In Figure 88 we can see in both 3D and 2D view of the planned trajectory going above the dynamic no-flight zone predicted for the real airplane.

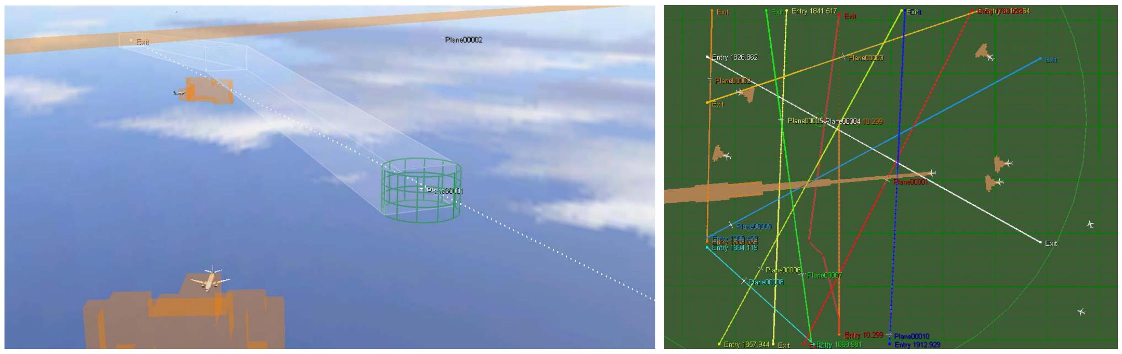


Figure 88: Operation of agent-controlled UAAs over LA with imported real air-traffic

19 Complex Combat Scenario

The scenario presented in this section utilizes all features provided by AGENTFLY system in the one complex scenario. All technology is combined together to provide the desired function in synergy. All airplanes in the scenario are controlled by the agents. Multi-agent system utilizes its flight path planning (Section 6) and simulation (Section 5) capabilities in the real-time simulation mode (see the Section 3). The airplanes which are normally driven by a human pilots use the multi-layer collision avoidance framework (Section 13) to avoid collisions during the mission. They use the cooperative collision avoidance algorithms (Section 14) to avoid collision among airplanes from their group but at the same time they need to avoid Global Hawks providing monitoring using the non-cooperative avoidance (Section 15). The Global Hawks are configured to not to use any collision avoidance module, they are fulfilling pre-configured mission objectives. The airplanes are divided into two packages A and B. They need to accomplish their mission targets and need to be coordinated together to fly in a formation package via specified corridors. This functionality is provided by the collective flight feature of the AGENTFLY system described in the Section 16. The situation of the coordination is made more difficulty by the need of pre-action and post-action refueling where occurs delays in the particular mission execution.

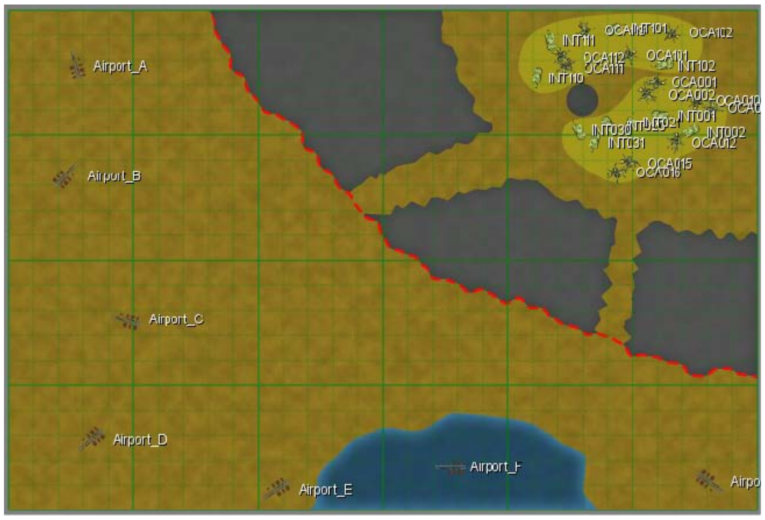


Figure 89: The map of the complex combat scenario.

The Figure 89 shows the map of the combat scenario. On the left hand side of the figure, there is the safe zone. There are four base airports and another three base airports are located in the bottom of the figure. These are the departure basis where the airplanes start from and where they land. The *Airport F* represents a carrier on the ocean. On the right hand side, there is the adversary zone. The battle line is depicted in red dashed line. Both packages are supposed to use the ingress corridor to get to the target zone (the corridor in the middle of the picture) and to use the egress corridor to leave the enemy zone. The grey zones on the map represent the zones where the air defense threat is present - they are the no flight zones in our representation. Two ochre zones in the right hand top corner represent the zones where the Desired Mean Points of Impacts (DMPI) are located. DMPIs are represented by small icons of tanks and helicopters in the map. The scenario simulate more than hour of the mission.

In the very beginning of the scenario at time 2 minutes after the start, the strike mission consisting of four *F16C*s departures from the *Airport G* and it aims to the *Rendezvous point* for *ingress corridor*. During the flight, the tanking of the fuel is scheduled. The tanker is late by 5

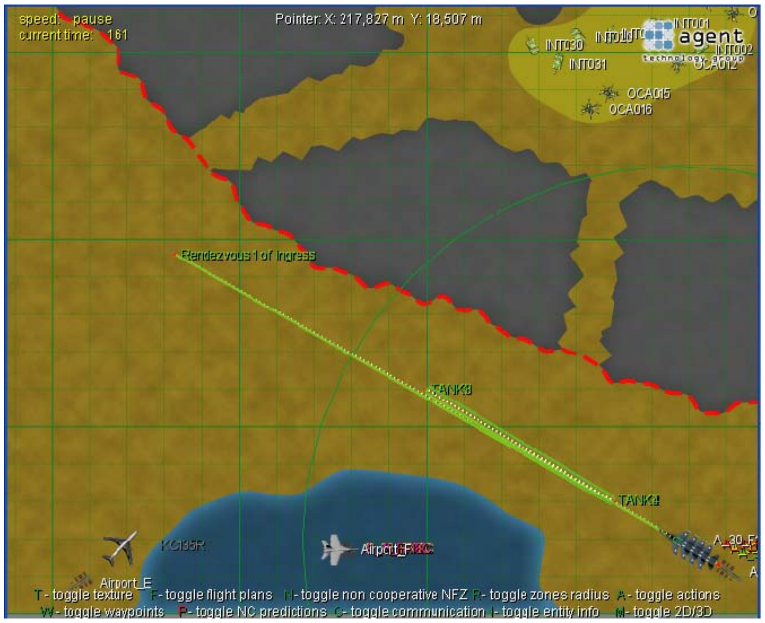

speed: pause
current time: 161
Pointer: X: 217,827 m Y: 18,507 m
Rendezvous 1 of Ingress
TANK9
TANK9
KC135R
Airport_F
Airport_E
T - toggle texture F - toggle flight plans N - toggle non cooperative NFZ R - toggle zones radius A - toggle actions
W - toggle waypoints P - toggle NC predictions C - toggle communication I - toggle entity info M - toggle 2D/3D

Figure 90: Pre-action refueling situation in the scenario.

minutes, therefore the mission must wait for the tanker. In the Figure 90, you can see the holding of the mission. The tanking process hasn't been implemented in detail in the AGENTFLY. The refueling process is represented only symbolically in the scenario and no refueling approach is realized in the scenario. When the *KC135R* comes, the mission follows *KC135R* closely. This represents the tanking process which causes the delay of the strike mission.

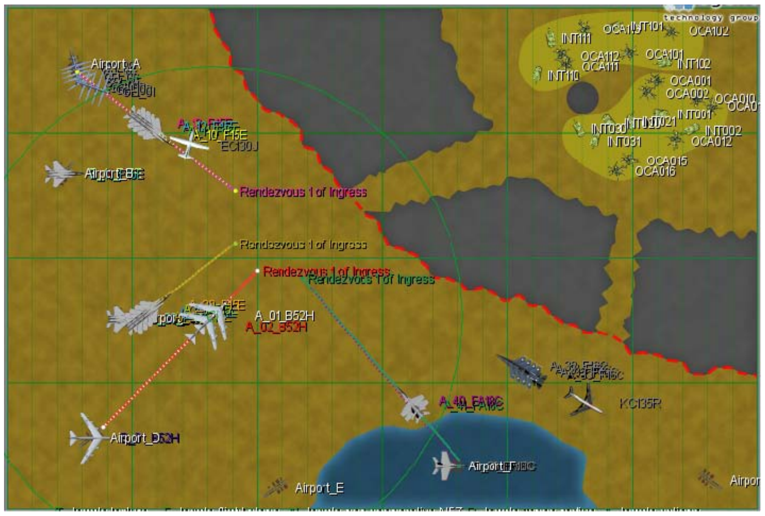


Figure 91: Members of the *package A* are flying to the *rendezvous point 1* of *ingress corridor* and the mission of four *F16C*s (blue ones) is delayed.

Around the time 9 minutes from start, the first *B52H* single-ship mission starts from the *Airport D*. The second single-ship mission starts at time 10 minutes and both missions aim to the *Rendezvous 1* for *Ingress corridor*, as they are going to be the members of the *package A*. Around the time 12 minutes, the *F15E* mission composed of 4 aircrafts starts from the *Airport A* and aims to the *Rendezvous 1* of *Ingress corridor* too. At the time 13 minutes after the start, the *FA18C* mission composed of 2 aircrafts starts from the carrier (*Airport F*) and aims to the same rendezvous point. Then another mission composed of four *F15E* aircrafts starts from the *Airport A*. Nine Global Hawks start just after the *F15E*s from the *Airport A*. They fly to the battle zone in order to monitor the situation. The situation around the time 15 minutes is in the Figure 91. During this phase the airplanes from the package A utilizes the automatic collision avoidance and performs initial inner group synchronization and monitoring.

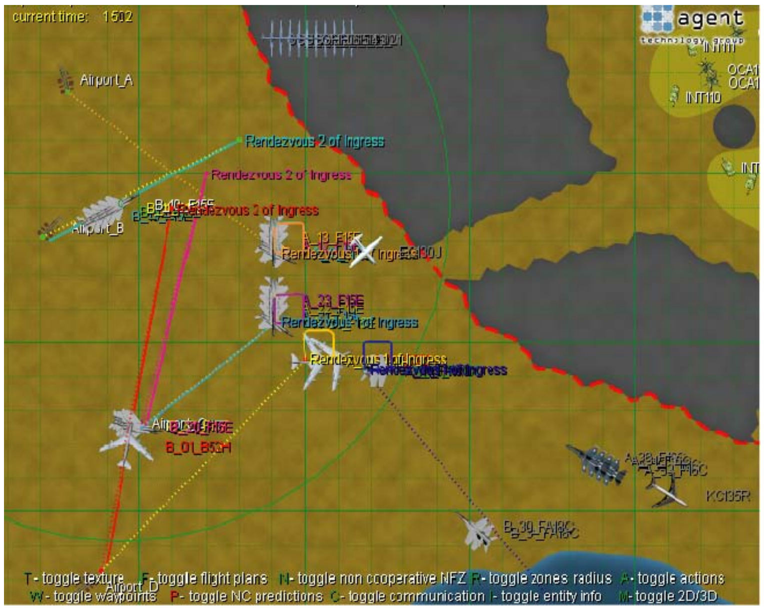

current time: 15:02
agent technology group
Airport_A
Rendezvous 2 of Ingress
Rendezvous 2 of Ingress
Rendezvous 1 of Ingress
Airport_B
Rendezvous 1 of Ingress
Rendezvous 1 of Ingress
Rendezvous 1 of Ingress
B_01_B52H
KC135R
INT110
OCA1
T - toggle texture F - toggle flight plans N - toggle non cooperative NFZ R - toggle zones radius A - toggle actions
W - toggle waypoints P - toggle NC predictions C - toggle communication I - toggle entity info M - toggle 2D/3D

Figure 92: *Package A* members are waiting in the holding orbits, *package B* members are flying to the *alternative rendezvous point 2*.

Around the time 19 minutes after the start, 4 missions of total 5 are at the *Rendezvous 1* for *Ingress corridor*. Using the group coordination they identified that the fifth mission is delayed because of the delayed tanker. Thus they are waiting in holding orbits for the last mission. During the waiting they are still doing monitoring of the fifth mission. If there is detected an issue with that mission (not enough fuel, other problems) they will decide to fulfill mission without last mission or cancel whole package plans.

At the time 20 minutes, the missions for the *package B* are taking off. The single aircraft mission *B52H* starts from the *Airport B*. First, the second package tries to go to the same *rendezvous location* as the first package. But when at least one mission of that package is closer (2 minutes later) to that point they detects that the desired contact point is already occupied by the another package. In such situation the second package decides to select alternative rendezvous point for the *Ingress corridor*. They select alternative area due to the fact that their priority (or order in the plan) is lower than the previous package. All members of *package B* will go the selected *Rendezvous 2*.

Later the mission composed of four *F15E*s starts from the *Airport B* and flies to the *Rendezvous 2* for ingress. So does the mission of two *F18C*, that starts from the *Airport F*. The described situation is depicted in the Figure 92. The members of the *package A* are waiting for the delayed mission at the *Rendezvous 1*. The delayed mission has just finished the refueling and flies to the approaching point. Meanwhile the second *package B* flies to the alternative *rendezvous point 2*.

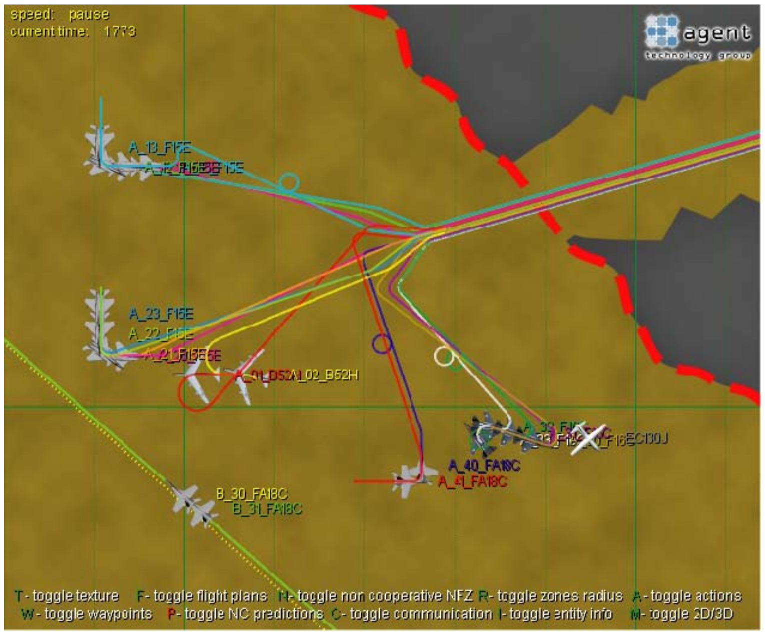

speed: pause
current time: 1773
agent technology group
A_13_F15E
A_23_F15E
A_22_F15E
A_21_F15E
A_01_B52H
A_02_B52H
A_40_FA18C
A_41_FA18C
B_30_FA18C
B_31_FA18C
EC130J
T - toggle texture F - toggle flight plans N - toggle non cooperative NFZ R - toggle zones radius A - toggle actions
W - toggle waypoints P - toggle NC predictions C - toggle communication I - toggle entity info M - toggle 2D/3D

Figure 93: Detail of the formation approach flight plans of the *package A*.

Around the time 29 minutes after the start the last mission of the *package A* finally comes to the rendezvous point and the formation is being formed. They negotiate about the selection of the position in the formation pattern and finds composition flight plans which are collision free. The final flight plans for the compilation formation members in to one package are shown in the Figure 93.

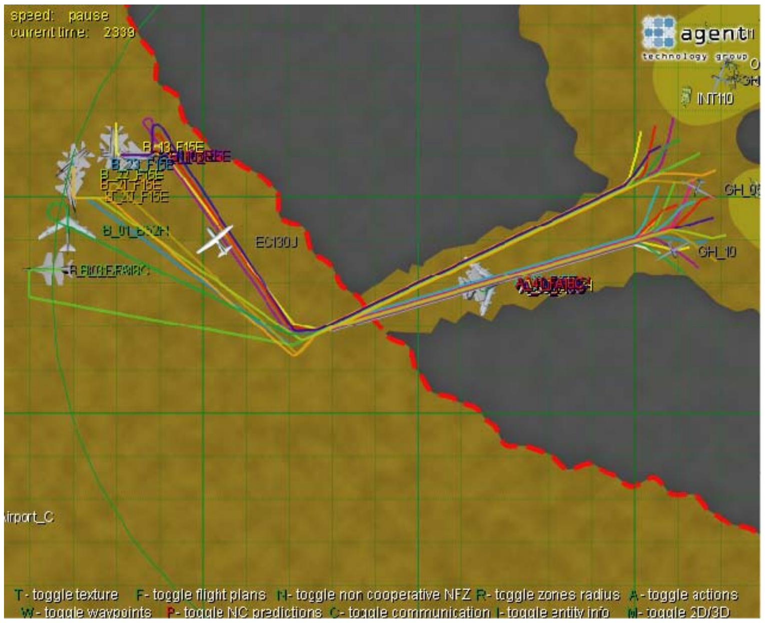

speed: pause
current time: 2339
agent technology group
INT110
GH_05
GH_10
B_13_F15E
B_23_F15E
B_22_F15E
B_21_F15E
B_20_F15E
B_01_B52H
EC130J
Airport_C
T - toggle texture F - toggle flight plans N - toggle non cooperative NFZ R - toggle zones radius A - toggle actions
W - toggle waypoints P - toggle NC predictions C - toggle communication I - toggle entity info M - toggle 2D/3D

Figure 94: *Package A* is in the *Ingress corridor* and the *package B* just finalized negotiation about formation setup.

Around the time 39 minutes, the *package B* starts the assemblage of the formation. The *package A* is still in the ingress corridor to the battle zone, see the Figure 94.

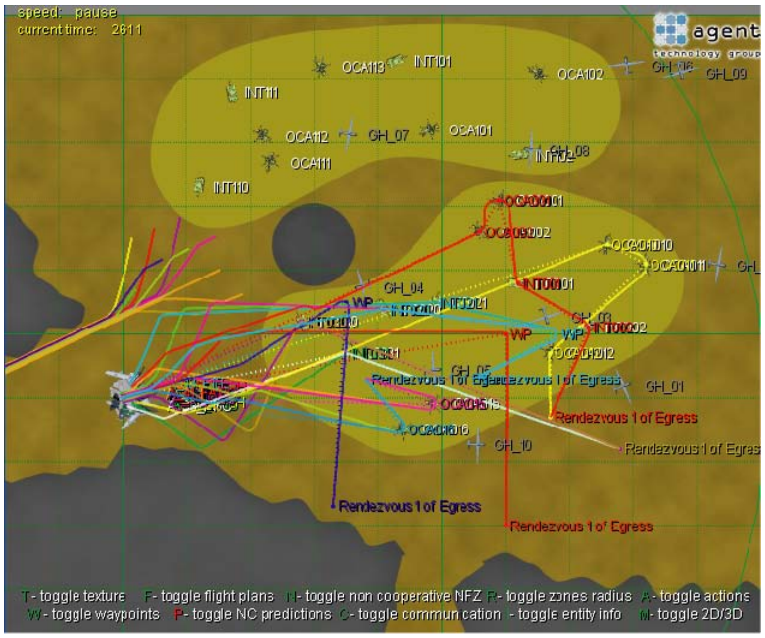

speed: pause
current time: 2611
agent technology group
Rendezvous 1 of Egress
T - toggle texture F - toggle flight plans N - toggle non cooperative NFZ R - toggle zones radius A - toggle actions
W - toggle waypoints P - toggle NC predictions C - toggle communication - toggle entity info M - toggle 2D/3D

Figure 95: Break up of the *package A*.

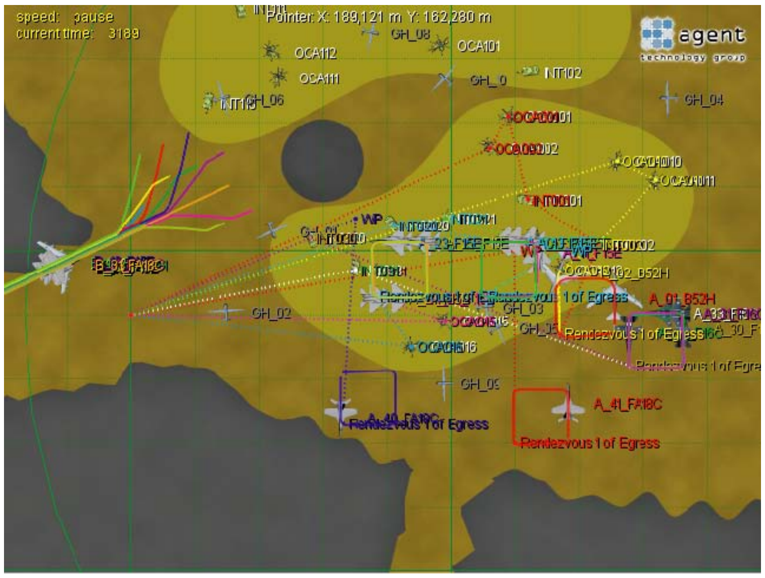


Figure 96: *Package A* members in the holding orbits waiting for all members to come to the rendezvous point of the *egress corridor*.

At the time 43 minutes, the formation of the *package A* breaks up and the individual airplanes fly to their DMPIs, see Figure 95. DMPIs of the package A are in the south part of the battle zone, under the grey no-flight zone. The airplane fly over all DMPIs on their lists and then they fly to the egress corridor, where is the rendezvous point. Meanwhile the Global Hawks monitor the situation around and the package members must avoid them using the non-cooperative collision avoidance.

When *package A* members fulfill their DMPIs, the *package B* members are flying through the *ingress corridor* to the battle zone.

A soon as all airplanes accomplish their objectives, they fly to the rendezvous point of *Egress corridor*, see Figure 96. Two *F18C* are patrolling in the holding orbits near the *egress corridor* to ensure the safety for other package members. In the same time, the *package B* is approaching the battle zone.

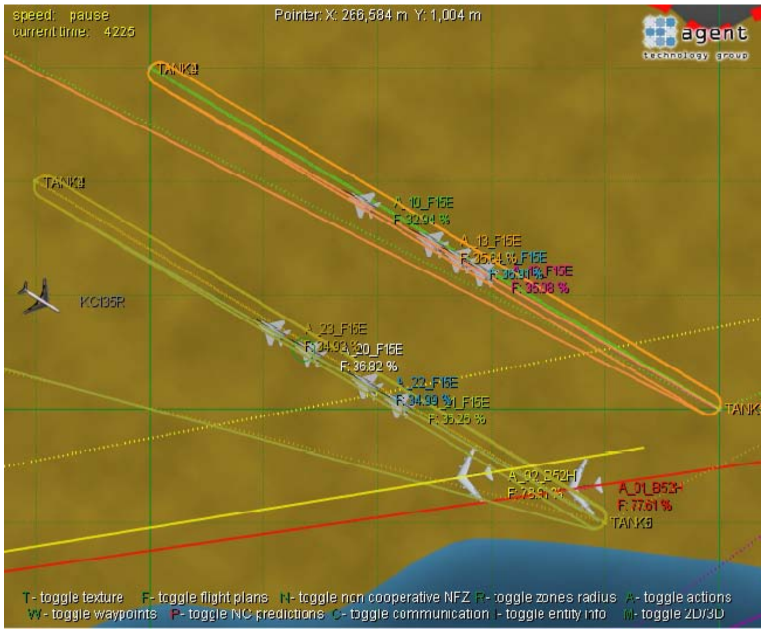

speed: pause
current time: 4225
Pointer: X: 266,584 m Y: 1,004 m
agent technology group
TANK4
TANK3
KC135R
A_10_F15E
F: 32.94 %
A_13_F15E
F: 25.64 %
F15E
F: 35.08 %
A_23_F15E
A_20_F15E
F: 36.92 %
A_22_F15E
F: 34.99 %
F15E
F: 35.25 %
A_02_B52H
A_01_B52H
F: 77.61 %
TANK5
TANK
T - toggle texture F - toggle flight plans N - toggle non cooperative NFZ R - toggle zones radius A - toggle actions
W - toggle waypoints P - toggle NC predictions C - toggle communication - toggle entity info M - toggle 2D/3D

Figure 97: Post-action tanking of the two missions of *F15E*s after the egress from the corridor.

After the egress formation assemblage, the *package A* leaves the battle zone using the *egress corridor* and the *package B* breaks up. The members of the *package B* fly to the north zone to their DMPIs. Around the time 65 minutes, the *package A* breaks up, because it has flown through the egress corridor over the battle line. The aircrafts form the same missions like before the ingress to the battle zone and they fly to their home bases. Both missions of *F15E*, each composed of 4 airplanes, are out of fuel. They have to meet the *KC135R* before their flight to their base. The tanking is again simulated by the flight behind the *KC135R*, see the Figure 97.

At the time 67 minutes, the **package B** assembles the formation and enters the *egress corridor*. At the time 80 minutes, the *package B* crosses the battle line and it breaks up. The missions are formed in the same way as before the *ingress* and the airplanes fly back home. In the same time, the Global Hawks fly back as well. The situation is depicted in the Figure 98. The two *F15E* missions are still holding a loop for the tanking.

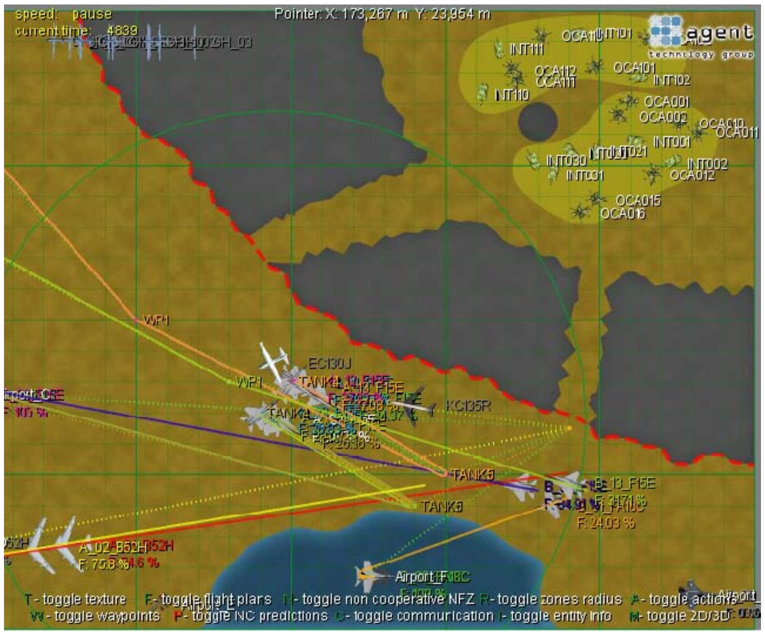

speed: pause
current time: 4839
Pointer: X: 173,267 m Y: 23,954 m
agent technology group
EC130J
KC135R
TANK5
TANK6
WP1
T - toggle texture F - toggle flight plans N - toggle non cooperative NFZ R - toggle zones radius A - toggle actions
W - toggle waypoints P - toggle NC predictions C - toggle communication I - toggle entity info M - toggle 2D/3D

Figure 98: The break up of the *package B* and flight to the home bases.

20 AGENTFLY Prototype Requirements

This section provides optimal requirements for running the basic demonstration scenarios of designed AGENTFLY prototype for all its components.

System Requirements: Multi-agent AGENTFLY Core System

One or more host computers with the following requirements are required to run the AGENTFLY Core. Use of several computers allows higher number of collision-free airways to be planned because planning is handled in a distributed manner based on plane-to-plane negotiation.

- JAVA supported operating system with 1GB RAM at least, CPU 1GHz+ (depends on the number of simulated UAAs),
- Java Runtime Environment 1.6 or higher (version for Windows 32bit is pre-installed on the distribution disk),
- network connection between all computers running Agents Core System (not necessary if the whole system is running on the one host) **without any firewall restrictions**.

System Requirements: Remote WEB Client

- standard PC with network connection to the AGENTFLY Core host (intranet/internet),
- Windows 32bit, Linux, Mac OS X, Sun OS (sparc or x86), 512 MB RAM at least,
- graphics drivers with OpenGL support,
- pre-installed Java Web Start (Java Runtime Environment 1.6 or higher is optional because it can be downloaded and installed by Java Web Start automatically),
- any internet browser associating the Java Web Start application with JNLP extension.

System Requirements: Operator Agent

Operator Agent can be started on the same computer as the Multi-agent AGENTFLY Core system part or on the other computer.

- Windows 32bit operating system, CPU 1GHz+, RAM 512MB+,
- graphic card with hardware 2D/3D acceleration supported in the drivers,
- minimal screen resolution 1024x768 pixels,
- Java Runtime Environment 1.6 or higher (version for Windows 32bit is pre-installed on the distribution disk),
- LAN network connection to the Core System host (only necessary when operator agent is running on different host).

References

[1] A-globe. A-globe Agent Platform. `http://agents.felk.cvut.cz/aglobe`, 2006.

[2] DOD. Unmanned aircraft systems roadmap 2005-2030, 2005.

[3] S. Frisken and R. Perry. Simple and efficient traversal methods for quadtrees and octrees. *Journal of Graphics Tools*, 7(3), 2002.

[4] Su-Cheol Han and Hyochong Bang. Proportional navigation-based optimal collision avoidance for uavs. In S. C. Mukhopadhyay and G. Sen Gupta, editors, *Second International Conference on Autonomous Robots and Agents*, pages 76–81. Massey University, New Zealand, 2004.

[5] JOGL. Java Bindings for OpenGL. `http://jogl.dev.java.net`, 2005.

[6] M. Naor. Bit commitment using pseudorandomness. *Journal of Cryptology: the journal of the International Association for Cryptologic Research*, 4(2):151–158, 1991.

[7] Simon Parsons and Michael Wooldridge. Game theory and decision theory in multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 5(3):243–254, 2002.

[8] M. Pěchouček, V. Mařík, D. Šišlák, M. Rehák, J. Lažanský, and J. Tožička. Inaccessibility in multi-agent systems. final report to Air Force Research Laboratory AFRL/EORD research contract (FA8655-02-M-4057), 2004.

[9] M. Pěchouček, P. Volf, D. Šišlák, and Š. Kopřiva. Project extension of the FA8655-06-1-3073 contract: Final report January 2008. Final report to Air Force Research Laboratory AFRL/EORD research contract extension, January 2008.

[10] Michal Pěchouček Přemysl Volf, David Šišlák and Magdalena Prokopová. Convergence of peer-to-peer collision avoidance among unmanned aerial vehicles. In *2007 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT 2007)*, pages 377–383, Silicon Valley, November 2007.

[11] Jeffrey S. Rosenschein and Gilad Zlotkin. *Rules of Encounter.* The MIT Press, Cambridge, Massachusetts, 1994.

[12] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall Series in Artificial Intelligence, Englewood Cliffs, New Jersey, 1995.

[13] David Šišlák, Martin Rehák, Michal Pěchouček, Milan Rollo, and Dušan Pavlíček. **$\mathcal{A}$-globe**: Agent development platform with inaccessibility and mobility support. In Rainer Unland, Matthias Klusch, and Monique Calisti, editors, *Software Agent-Based Applications, Platforms and Development Kits*, pages 21–46, Berlin, 2005. Birkhauser Verlag.

[14] M. Wooldridge, editor. *An Introduction to MultiAgent Systems.* John Wiley and Sons Ltd, 2002.

[15] Gilad Zlotkin and Jeffrey S. Rosenschein. Negotiation and task sharing among autonomous agents in cooperative domains. In N. S. Sridharan, editor, *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 912–917, San Mateo, CA, 1989. Morgan Kaufmann.